University of Alberta

Advanced Software Engineering Models for Quality Improvement

by

Milorad Stefanovic

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2001

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-60501-9

Canada

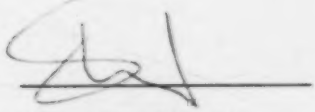
University of Alberta

Library Release Form

Name of Author:	Milorad Stefanovic
Title of Thesis:	Advanced Software Engineering Models for Quality Improvement
Degree:	Master of Science
Year this Degree Granted:	2001

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material from whatever without the author's prior written permission.



Milorad Stefanovic

Apt. 404, 12915 65th Street

Edmonton, Alberta

Canada T5A 0Z8

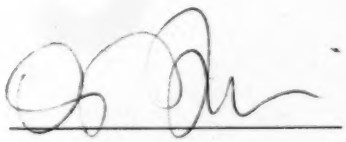
Submitted to the Faculty of Graduate Studies and Research:

MARCH 6TH 2001

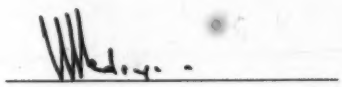
University of Alberta

Faculty of Graduate Studies and Research

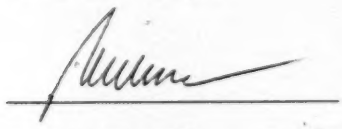
The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Advanced Software Engineering Models for Quality Improvement submitted by Milorad Stefanovic in partial fulfillment of the requirements for the degree of Master of Science



Dr. Giancarlo Succi



Dr. Witold Pedrycz



Dr. Petr Musilek

Dr. Eleni Stroulia

Thesis approved by committee:

MARCH 6TH 2007

Abstract

Software projects often suffer from low quality caused by a lack of control over the development process. Software metrics and models are invaluable for process characterization and quality improvement.

The impact of the object-oriented design choices on the quality of the resulting system is investigated and innovative statistical methods are applied to deal with the peculiarities of the software engineering data. The ability of these models is empirically validated using industrial datasets, identifying approximately 50% of the classes causing 80% of the defects in the system.

Software reliability growth models are adopted to characterize the occurrence of service requests and help in resource allocation using multimodel approach. Models' performance is assessed with respect to specific goals, using combined numerical and graphical methods. Parametric and non-parametric statistical methods are used to fully describe this process and sensitivity of the models to the imprecision of the input data.

Table of Contents

1.Introduction _____ 1

1.1. Software Engineering Discipline _____ 4

1.2. Software Development Process _____ 5

1.3. Costs of Software Development _____ 8

1.4. Object-Oriented Design _____ 9

1.5. Software Validation _____ 9

1.6. Managing and Scheduling Software Projects _____ 10

2. Software Quality and Reliability _____ 11

2.1. Defects, Failures, and Service Requests _____ 12

3. Software Metrics _____ 14

3.1. Types of Software Metrics _____ 15

3.2. Structural Metrics _____ 16

3.3. Object-Oriented Metrics _____ 18

3.4. Validity of Software Metrics _____ 19

3.5. The Goal Question Metric Paradigm _____ 20

3.6. Types of Metrics Data _____ 22

3.7. Parametric and Nonparametric Statistics _____ 26

3.8. Pragmatic Approach _____ 27

4. Data Analysis in Software Engineering _____ 28

4.1. Types of Models in Software Engineering _____ 28

4.2. Problems with Software Engineering Data _____ 30

4.3. Regression Models _____ 33

4.3.1. Linear Regression Model _____ 34

4.3.2. Ordinal Least Squares and Maximum Likelihood Methods _____ 35

4.3.3. Poisson Regression Model _____ 36

4.3.4. Negative Binomial Regression Model ____ 37
4.3.5. Zero Inflated Regression Model ____ 41
4.4. Software Reliability Growth Models ____ 42
4.5. Gamma Analysis ____ 44
4.6. Methods and Criteria for Models Comparison ____ 45
4.6.1. Correlation Coefficient ____ 46
4.6.2. Alberg Diagrams ____ 47
4.6.3. Sensitivity Analysis of the Models ____ 48
5. Empirical Investigation in Software Engineering ____ 49
5.1. Basic Research Techniques ____ 49
5.2. Framework for Empirical Investigation ____ 51
5.3. Data Collection Process ____ 53
5.3.1. Metrics Extraction ____ 54
5.3.2. WebMetrics Relations Language ____ 56
5.3.3. The CK Metrics Example ____ 57
6. Analysis of Service Requests ____ 59
6.1. Discussion of the Experimental Data from Real-time Domain ____ 59
6.1.1. Extraction and Analysis of the Models for Timing of SRs ____ 63
6.1.2. Time to Resolve the SRs ____ 67
6.1.3. Kinds of SRs ____ 69
6.1.4. Sensitivity of the Models ____ 70
6.1.5. Summary of the Results and Discussion ____ 72
6.2. Discussion of the Experimental Data from Commercial Domain ____ 74
6.2.1. Extraction and Analysis of the Models for timing of SRs ____ 77
6.2.2. Time to Resolve the SRs ____ 82
6.2.3. Kinds of SRs ____ 83
6.2.4. Sensitivity of the Models ____ 85

6.2.5. Summary of the Results and Discussion ____ 87
7. Impact of Object-Oriented Design on Class Defect Behavior ____ 90
7.1. Discussion of the Experimental Data from Real-Time Domain ____ 90
7.1.1. Extraction of the Models ____ 94
7.1.2. Analysis of the Results ____ 98
7.2. Discussion of the Experimental data from Commercial Domain ____ 102
7.2.1. Extraction of the Models ____ 105
7.2.2. Analysis of the Results ____ 107
7.3. Related Studies and Discussion ____ 112
8. Conclusion ____ 115

List of Tables

Table 1: GQM framework used in this study ____ 22

Table 2: Scale types (taken in part and with modifications from Fenton and Pfleeger, 1997) ____ 23

Table 3: Software reliability growth models (adopted version of Table A in Wood, 1996) ____ 43

Table 4: Metrics collected by WebMetrics ____ 54

Table 5: Set of WebMetrics relations ____ 57

Table 6: Expression of CK metrics in terms of relations ____ 58

Table 7: Severity levels of the SRs in the four projects ____ 61

Table 8: Types of SRs in projects A and B ____ 62

Table 9: Result of the analysis of the SRGMs ____ 65

Table 10: Ranking of the SRGMs ____ 66

Table 11: Coefficients of linear regressions ____ 68

Table 12: Ranking of the SRGMs with respect to their sensitivity to error ____ 72

Table 13: Severity and priority levels of the SRs in the P1 dataset ____ 75

Table 14: Types of SRs in P1 dataset ____ 75

Table 15: Result of the analysis of the SRGMs for the three projects ____ 80

Table 16: Ranking of the SRGMs for the three projects ____ 81

Table 17: Coefficients of linear regression ____ 83

Table 18: Ranking of the SRGMs with respect to the robustness to noise in the input data ____ 87

Table 19: Comparison of the results in real-time and commercial domain ____ 90

Table 20: Number of available data points for projects ____ 91

Table 21: Descriptive Statistics of the extracted metrics and defects for the projects ____ 92

Table 22: Correlations between number of defects and internal metrics for the projects _____ 95

Table 23: Performance of the models with respect to correlation and dispersion 97

Table 24: Median values of RFC _____ 100

Table 25: Percentile of classes identified for inspection to detect 80% of the defects _____ 102

Table 26: Descriptive Statistics of the extracted CK metrics and LOC for project A _____ 103

Table 27: Correlation coefficients between the extracted metrics and number of defects _____ 104

Table 28: Coefficients of the PRM with statistical significance _____ 106

Table 29: Resulting models _____ 107

Table 30: Ranking of the models with respect to the applied criteria _____ 112

Table 31: Related studies _____ 114

List of Figures

Figure 1: An example of cost distribution in software development _____ 8

Figure 2: Poisson and NB distribution for μ=3 _____ 40

Figure 3: An example of Alberg diagram _____ 48

Figure 4: Framework for empirical investigation in software engineering _____ 53

Figure 5: Relations as additional abstraction layer in software metrics analysis 54

Figure 6: Temporal evolution of the four projects _____ 59

Figure 7: Arrival time of Service Requests for the four projects _____ 60

Figure 8: Scatterplot of the variation of the time to serve with the regression line _____ 67

Figure 9: Gamma analysis of the severity _____ 70

Figure 10: Gamma analysis for projects A and B _____ 70

Figure 11: Performance of SRGMs in presence of error _____ 71

Figure 12: Performances in presence of noise of (a) the linear regression models and (b) the gamma analysis _____ 72

Figure 13: Occurrence of SRs over time for the three projects _____ 78

Figure 14: Scatterplot of the coverage of fit vs. relative precision of fit (a), and the zoomed portion of this graph (b) _____ 82

Figure 15: Variation of time to service SRs with calendar time _____ 83

Figure 16: Severity and priority levels of SRs for the P1 dataset _____ 84

Figure 17: Gamma analysis for P1 with respect to the severity, priority level, and types of SRs _____ 85

Figure 18: Monte Carlo analysis of the robustness of SRGMs to the noise in the input data _____ 86

Figure 19: Robustness of the gamma analysis linear regression results in presence of the noise in the input data _____ 87

Figure 20: Boxplots for distribution of the number of defects _____ 93

Figure 21: Cumulative number of defects for the five projects _____ 94

Figure 22: RFC boxplots for the five projects_____ 99

Figure 23: Alberg diagrams for the RFC-based models _____ 100

Figure 24:Histogram of the number of defects _____ 104

Figure 25: Histograms of RFC and DIT _____ 108

Figure 26: Alberg diagrams for the models and predicted percentiles of the classes with 80% of the defects in the system_____ 110

1. Introduction

Studies show that, on the average, software development projects overshoot their schedule by half and larger projects generally do worse (Gibbs, 1994; Brooks, 1987; Pressman, 1996). For every six new large software systems which are put into operation, two others are canceled. It is reported that three quarters of all large systems are "operating failures" that either do not function as intended or are not used at all.

Software projects fail due to lack of control over the software development process. Quality is a key element in success of any software product. Assuring high quality is an increasingly complex time- and effort-consuming activity. A proper characterization and understanding of this process is thus essential in achieving higher quality software. Software engineering uses a systematic approach to the development of software. Models based on software metrics provide a way of quantitative management of software quality.

A great part of effort and time in the development of a software product is spent on servicing demands for modifications in the behavior of the system (Basili and Weiss, 1984). The demands for modification of the system are referred to as Service Requests (SRs). Due to the high impact SRs have on the overall process of software development, time to market, and the customer's satisfaction (Bays, 1999), there is clearly a need for further investigation to define a systematic and replicable framework for the analysis of SRs.

Design of an object-oriented software system offers a substantial amount of information about the software system even before any coding has started. A widely used set of software metrics for object-oriented systems is the suite proposed by Chidamber and Kemerer (1994). This set of metrics, referred to as the CK suite, measures different aspects of software design, such as complexity, coupling, cohesion, and inheritance. Clearly, it is valuable to empirically validate the ability of these metrics to be used for identification of the critical classes that consume most of the development effort and resources and need special attention in the development and testing activities.

For the projects analyzed in this study, all the modifications of the classes caused by defects in software operation are recorded throughout the development process. It is assumed that the number of modifications referring to defects represents a good estimation of the defect-proneness of the class (Fenton and Pfleeger, 1997).

This study proposes novel methods for prediction and description of defect behavior of the classes in object-oriented systems, and investigates possibilities for better resource allocation in the process servicing requests for modifications in such systems. The presented models quantify the impact of choices made in object-oriented design on the quality of the resulting system. Furthermore, the existing and new software reliability growth models are adopted to characterize the timings of service requests and help in resource allocation.

Innovative statistical methods are applied to deal with the peculiarities of the software engineering data, such as non-normally distribution, overdispersed count data on the absolute measurement scale, and high occurrence of zero counts. These models are employed for estimation of the number of defects from the design metrics (class 3 model; Fenton and Pfleeger, 1997). The ability of Poisson regression model, negative binomial regression model, and zero-inflated negative binomial regression model to identify defect-prone classes in the system is validated on different industrial datasets from industrial environment in real-time telecommunication and commercial application domains.

In combination with classical statistical methods, such as correlation coefficients and standard error, multiple combined methods are used for assessment of models' performance with respect to specific goals, including dispersion coefficient and graphical method referred to as Alberg diagram.

The results of the analysis performed in this study show that the proposed methodology is both feasible and useful. Although the presented results are specific for the projects developed in these environments, the process used in the analysis can be easily replicated and applied in different environments.

The models applied in this study successfully explain the excessive variability in the data and can be used to identify approximately 50% of the classes causing 80% of the defects in the system. Measure of communication between classes, i.e., response for a class, shows a good potential as a predictor for this purpose over the projects analyzed in this study.

One of the goals of this thesis is to help project managers in making informed decisions in the servicing process. The appropriate characterization and understanding of this expensive and time-consuming activity enables better planning and scheduling, resource allocation, identification of bottlenecks, and accurate estimation of the time required for a software system to become stable (Basili *et al.*, 1996). Based on this information, different projects and development processes can be assessed and compared with respect to the process of servicing SRs.

The framework for an accurate description of the various aspects of SRs occurrence is proposed in this study. Software reliability growth models are adopted to characterize the occurrence of service requests and help in resource allocation using multimodel approach. Proposed framework provides a quantitative comparison and assessment of the descriptive power of the different models applied to occurrence of SRs. Models' performance is assessed with respect to specific goals, using combined numerical and graphical methods.

In addition to the software reliability growth models, gamma analysis and parametric linear regression are used to fully describe the occurrences of service requests over time. A Monte Carlo simulation is performed to assess the sensitivity of the proposed models to the imprecision of the input data, typical when people participate in the data collection process. Results indicate that a multimodel approach is recommended for most reliable results in analysis of service requests.

The thesis is organized as follows: The rest of Section 1 provides a background overview of the software engineering discipline and software development processes. Issues specific to software quality and reliability are discussed in

Section 2. A systematic summary of the software metrics and associated statistical methods for measurement of different software attributes is provided in Section 3. A detailed overview of various data analysis techniques applicable to software engineering is provided in Section 4. This section also provides explanation of the software reliability growth models and innovative models for overdispersed count data with excess zeroes, such as the zero-inflated negative binomial model. The potential of these models to be used for quality improvement is empirically evaluated in later sections of the thesis. Empirical investigation techniques and data collection process are discussed in Section 5. The rest of the thesis is organized around the analyses of the specific datasets from different industrial software development environments. Analysis of software service requests for four projects from the real-time telecommunication domain and three datasets from two companies in commercial domain is described in Section 6. The impact of object-oriented design on the defect behavior is investigated in Section 7 using five real-time and two commercial projects. The naming scheme for the fourteen projects used in this study is based on the corresponding sections where the particular projects are analyzed. Discussion of the results and conclusion are provided in Section 8.

1.1. Software Engineering Discipline

Virtually all domains of life now depend on complex computer-based systems. More and more products incorporate computers and controlling software in some form. The software in these systems represents a large and increasing proportion of the total system costs. Therefore, producing software in a cost-effective way is essential for the functioning of the international economy.

Software engineering is a discipline whose goal is the cost-effective development of software systems. As an engineering discipline, software engineering recognizes existing organizational and financial constraints and looks for solutions within these constraints. It is not just concerned with the technical processes of software development but also with activities such as software project management and methods to support software production. Software

engineering is concerned with all aspects of software production from early stages of system specification through to maintaining the system after its release.

Software is abstract and not constrained by materials, physical laws or manufacturing processes. This lack of limitations simplifies parts of software engineering, giving it great potentials. On the other hand, software easily becomes extremely complex and difficult to understand.

Software engineering is a relatively young discipline, first proposed in 1968 (Sommerville, 2001). It emerged from early experience that an informal approach to software development was not enough for building complex software applications. This problem was referred to as "software crisis" (Gibbs, 1994). Software projects were typically late, unreliable, difficult to maintain, costing more than predicted. It was clear that new techniques and methods were needed to control the complexity of such complex systems.

Some of the software engineering techniques have become part of software engineering and are now widely used. However, there are still problems in producing complex software which meets user expectations, is delivered on time and to budget. Many software projects still have problems and this has led to some commentators (Pressman, 1996) suggesting that software engineering is in a state of chronic affliction.

As the ability to produce software has increased so too has the complexity of the software systems required. New technologies and requirements in different application domains place new demands on software engineering. There is a constant need and room for improvement.

1.2. Software Development Process

A software development process is defined as a set of all activities that produce a software product (Sommerville, 2001).

Software development processes are complex and heavily dependent on human judgment and creativity. Although there is a wide variety of different software

processes, there are four basic activities common to all software processes: specification, development, validation, and evolution.

Software specification activity defines the functionality of the software and constraints on its operation.

Software development activity is focused on producing the software to meet the specifications.

Software validation activity ensures that the software does what the customer wants.

Software evolution is concerned with the changes that have to be made to software to meet changing customer needs.

Different software processes organize these activities in different ways and are described at different levels of detail. The timing of the activities varies, as does the result of each activity. Different organizations may use different processes to produce the same type of product. However, some processes are more suitable than others for some types of application. If an inappropriate process is used, this will probably reduce the quality or the usefulness of the software product to be developed.

Although there is no 'ideal' software process, there is a lot of scope for improving the software process in many organizations. Many organizations still rely on ad hoc processes and do not take advantage of software engineering methods in their software development.

A software process model is a simplified description of a software process which is presented from a particular perspective. Software process model is a simplification of the actual process which is being described. Process models may include activities which are part of the process, software products, and the personnel and resources involved in software engineering.

There are a number of different general models or paradigms of software process, such as the waterfall, evolutionary, and component-based approaches.

The waterfall approach takes the software development activities and represents them as a separate process phases, such as requirements specification, software design, implementation, testing, and evolution. After each stage is defined the development process goes on to the following stage.

The evolutionary development approach interleaves the activities of specification, development, and validation. An initial system is rapidly developed from very abstract specifications. This is then refined with customer input to produce a system which satisfies the customer's needs. The system may then be delivered. Alternatively, it may be reimplemented using a more structural approach to produce a more robust and maintainable system.

The component approach to software development, based on assembly from reusable components, assumes that parts of the system already exist. The system development process focuses on integrating these parts rather than developing them from scratch.

Some software processes are more mature than others, as noted by the Software Engineering Institute's (SEI) reports on process maturity (Humphrey, 1989; Paulk, 1991). While some organizations have clearly-defined processes, others are more volatile, changing significantly with the people who work on the projects. The SEI has suggested that there are five levels of process maturity, ranging from ad hoc (the least controllable) to repeatable, defined, managed, and optimizing (the most controllable).

SEI Capability Maturity Model (CMM) uses process visibility as the key discriminator among a set of maturity levels. The more visibility into the overall development process, the higher the maturity and the better managers and developers can understand and control their activities. Section 3 provides more details on use of software metrics to increase visibility and control over the development process.

1.3. Costs of Software Development

The goal of software engineering is Software production of high-quality software in a cost-effective way (Sommerville, 2001).

There is no simple description of the software development costs. The precise distribution of costs across the software process depends on a lot of different factors, including the process used and the type of software that is being developed.

The percentage distribution of the total cost of developing a typical software system can be roughly represented with Figure 1. However, the distribution of cost greatly depends on the development process, application domain, programming language etc.

If the software is developed using an evolutionary approach, there is no clear distinction between specification, design, and development. Specification, design, implementation, and unit testing are carried out in parallel within a development activity. However, there is still a need for a separate system testing activity once the initial implementation is completed.

Overall, system integration and testing tend to be the most expensive development activity. For critical systems, this phase can account for 50% of the total system costs. This study focuses on this aspect of software development through the analysis of the process of servicing requests for change in the software system (Section 6), and through the analysis of the design characteristics with most significant impact on defect behavior of the classes in an object-oriented system (Section 7).

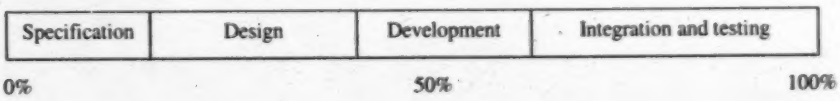


Figure 1: An example of cost distribution in software development

On the top of development costs, costs are also incurred in changing the software after it has been released and gone into use. For many software systems which have a long lifetime, these costs are likely to exceed the development costs by

factor of 3 or 4. The analysis of service requests performed in this study provides an objective support for determination when the product is ready to be released.

1.4. Object-Oriented Design

Although software design is still an *ad hoc* process in many software projects, there is a clear need for formal design management and change control.

'Structured methods' propose a more structured approach to software design. Structured methods represent sets of notations and guidelines for software design. Examples of include Structured Design (Constantine and Yourdon, 1979), Jackson System Development (Jackson, 1992), and various approaches to object-oriented design (Booch, 1994; Rumbaugh et al., 1991, Jacobson, 1992).

In addition to classical structured, top-down design methods, there is a need for a software development paradigm that can handle the added complexity and complex user interfaces. One of the biggest reasons for moving to the object-oriented paradigm for developing complex applications is that it allows designers to model the real world more closely (Riel, 1996).

Object-oriented methods include an inheritance model of the system, models of the static and dynamic relationships between objects and model of objects interactions at run time. The analysis of design issues of the software systems in this study is performed using the CK suite object-oriented design metrics.

Various structured methods have been applied successfully in many large projects. The success of methods often depends on their suitability for an application domain and specific organization. This study focuses on empirical data from industrial environment in telecommunication and commercial application domain.

1.5. Software Validation

The role of the software validation activity is to check if a software system conforms to specifications and if it meets customer's expectations. The majority

of validation effort is incurred in the testing phase. Validation also involves inspection and reviews at each stage of the software development process.

Complex software systems are built out of sub-systems, which are built out of modules. These systems are not tested as single, monolithic unit. Testing is carried out incrementally in conjunction with system implementation. Typical stages of the testing process are:

Unit testing – Individual components (modules), such as classes of an object-oriented system, are tested to ensure that they operate correctly. Each component is tested independently, without other system components.

Sub-system testing – This phase involves testing collections of modules which have been integrated into sub-systems. The most common problems which arise in large software systems are interface mismatches. The sub-system test process therefore concentrates on detection of module interface errors by rigorously exercising these interfaces.

System testing – The sub-systems are integrated to make up the system. This process is concerned with finding errors that result from unanticipated interactions between sub-systems. It is also concerned with validating that the system meets its functional and non-functional requirements.

Acceptance testing – This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than simulated test data. Acceptance testing may reveal errors and omissions in the system requirements definition because the real data exercise the system in different ways from the test data.

Various types of SRs are result of the software validation activities. The occurrence of SRs and its impact on allocation of effort and resources in the development process are discussed in Section 6 of this study.

1.6. Managing and Scheduling Software Projects

Software managers face a particularly demanding task of project scheduling. Managers estimate the time and resources required to complete activities and

organize them into a coherent sequence. Unless the project being scheduled is similar to a previous project, previous experience is an uncertain basis for new project scheduling. Schedule estimations is further complicated by the fact that different projects may use different design methods and implementation languages.

Project scheduling involves separating the total work involved in a project into separate activities and judging the time required to complete these activities.

As well as calendar time, managers must also estimate the resources needed to complete each task. The principal resource is the human effort required.

Software metrics and models provide a quantitative way to systematically build corporate experience and make informed estimates and decisions.

2. Software Quality and Reliability

Quality of a software product is influenced by the development technology, characteristics of the development process, and quality of the personnel involved in the development (Sommerville, 2001).

There has been a lot of effort to form a single model for expressing software quality. An international standard for software quality was proposed in 1992. This standard is called *Software Product Evaluation: Quality Characteristics and Guidelines for their Use*, but more commonly referred to as ISO 9126 (Ince, 1994). In the standard, software quality is defined to be: "The totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied need."

ISO 9126 decomposes quality into six factors: functionality, reliability, efficiency, usability, maintainability, and portability.

This decomposition approach to software quality measurement requires careful planning and data collection. Proper implementation even for a small number of quality attributes uses extra resources that managers are often reluctant to supply. A framework for empirical analysis of software quality is proposed in Section 5.2, and different aspects of quality are investigated in Sections 6 and 7.

Software reliability represents a key attribute of software quality. Quantitative methods for its assessment evolve from the theory of hardware reliability and date back to early 1970s (Goel and Okumoto, 1979).

The basic problem of reliability theory is to predict when a system will eventually fail. The same approach applies in software. Although there are many reasons for software to fail, none involves wear or tear. Usually, software fails because of a design problem. The impact of software design aspects on the defect behavior of the system is investigated in detail in Section 7 in this study.

Rather than maintaining reliability, as it is done with hardware systems, software reliability is growing over time. This aspect of software reliability is discussed in Section 4.4 and empirically validated using data from industrial environment in Section 6.

2.1. Defects, Failures, and Service Requests

It is important to measure different aspects of software quality in order to improve the overall quality of software systems. Such information can be useful to determining:

How many problems have been found with a product;

How effective are the prevention, detection, and removal process;

Whether the product is ready for release to the next development stage or to the customer;

How the current version of the product compares in quality with previous or competing versions.

The terminology used to support this investigation and analysis must be precise, allowing understanding the causes as well as effects of quality assessment and improvement efforts.

There is considerable disagreement about the definition of defects, errors, faults, and failures (Fenton and Neil, 1999). In different studies, defect counts refer to post release defects or defects discovered after some arbitrary point in the software cycle, and total known defects.

A fault occurs when a human error results in a mistake in some software product. For example, a developer might misunderstand a user-interface requirement, and therefore create a design that includes the misunderstanding. The design fault can also result in incorrect code, as well as incorrect instructions in the manual. Thus, a single error can result in one or more faults, and a fault can reside in any of the products of development.

On the other hand, a failure is the departure of a system from its required behavior. Failures can be discovered both before and after system delivery, as they occur in testing as well as in operation.

Faults represent problems that developer sees, while failures are problems that the user sees. Not every fault results in a failure, since the conditions under which a fault results in system failure may never be met.

The reliability of a software system is defined in terms of failures observed during operation, rather than in terms of faults. It is usually impossible to infer much about reliability from fault information alone. System containing many faults may be very reliable, because the conditions that trigger the faults may be very rare.

One of the problems is that the terminology is not uniform, including terms such as: errors, anomalies, defects, bugs, crashes, etc. There is no general consensus on what constitutes a defect. A defect can be either a fault discovered during review and testing, or a failure that has been observed during software operation.

A good, clear way of describing what is done in reaction to problems is also necessary. For example, if an investigation of a failure results in the detection of fault, then a change is made to the product to improve it. A change can also be made if a fault is detected during a review or inspection process. In fact, one fault can result in multiple changes to the product.

SRs, discussed in this study, represent a more general term that helps in answering the questions from beginning of this section. Multimodel approach for this analysis based on the software reliability growth models is presented in Section 6.

As already mentioned, the terminology differs widely between studies. In addition to defects and similar concepts, there is also a notion of a service request. A great part of effort and time in the development of a software product is spent on servicing demands for modifications in the behavior of the system. These demands are referred to as Service Requests (SRs). Due to the high impact SRs have on the overall process of software development, there is clearly a need for further investigation to define a systematic and replicable framework for the analysis of SRs. The process of servicing SRs requires careful control based on multiple goals, such as time to market and customers satisfaction (Bays, 1999).

Counts restricted only to defects have a limited power in predicting reliability because, despite usefulness from a system developer's point of view, they do not measure the quality of the system as the user is likely to experience it (Fenton and Neil, 1999). SRs, on the other hand, have higher potential in measuring the general quality of the product since they also capture requests for modifications triggered from customers in order to improve quality and usability of the product from their standpoint.

This kind of information is also very useful for product managers, helping them allocate available resources in best possible way in order to achieve their goals. The patterns in occurrence of SRs with respect to their severity and the type, occurrence over time and effort necessary fixing them are discussed in Section 6, and the appropriate models are built to support management in resource allocation decisions.

3. Software Metrics

Measurement offers visibility into the ways in which the processes, products, resources, methods, and techniques of software development relate to one another. It can help in answering questions about the effectiveness of techniques or tools, the productivity of development activities such as testing and configuration management, the productivity of products and more.

In addition, measurement is used to define a baseline for understanding the nature and impact of proposed changes. Finally, measurement allows managers and

developers to monitor the effects of activities and changes on all aspects of development, so that action can be taken as early as possible to control the final outcome.

3.1. Types of Software Metrics

Measurement is performed on software development process and the various software products (Basili and Rombach, 1988), and resources (Fenton and Pfleeger, 1997). Improving product requires proper characterization of the available resources in addition to understanding the product and the process.

Every measurement activity has to identify entities and attributes to be measured. In software metrics, there are three such classes: processes, products, and resources.

A process represents a collection of software-related activities. A product is any artifact that results form a process activity. Resources are entities required by a process activity.

Within each class of entity, there are internal and external attributes. Internal attributes of a product, process, or resource are those that can be measured purely in terms of the product, process, or resource itself, separate from its behavior.

External attributes of product, process, or resource are those that can be measured only with respect to how the product, process, or resource relates to its environment.

An example of a product attribute is the design. Some examples of internal aspects of design are size, coupling, complexity, and cohesiveness. External aspects are quality, complexity for use, and extendibility.

Examples of internal attributes of the testing process are time, effort, and number of discovered problems or defects, while external attributes are cost and cost-effectiveness.

Internal attributes of human resources in software development are effort and utilization, and external aspects are productivity, experience, and satisfaction.

Product metrics are concerned with characteristics of the software itself. There are two classes of these metrics: dynamic and static.

Dynamic metrics are collected by measurements made of a program in execution, while static metrics are collected by measurements made of the system representations such as design, code, or documentation.

These different types of metrics are related to different quality attributes. Dynamic metrics help to assess the efficiency whereas static metrics help to assess the complexity.

In addition to the objective metrics, subjective metrics are also required in some cases, especially for aspects such as experience of personnel, type of application, understandability etc. These aspects are typically categorized to a reasonable degree of accuracy on a nominal measurement scale.

The CK metrics suite collected in this study represents an example of objective static internal product metrics, while SRs represent external metrics. Some aspects of SRs, such as severity and priority are assigned subjectively on an ordinal scale.

3.2. Structural Metrics

The most obvious and easiest to understand internal software product attribute is the size of the software system. The size is a static attribute that can be measured without having to execute the system. However, even for this relatively simple attribute, there are multiple ways to measure it regarding to the different perspective of interest. For example, there are aspects of physical length and functionality for the user. The complexity and the amount of genuinely new software developer should also be taken into account.

There are three major products of the software development process whose size can be measured: specification, design, and code. The commonly used measure of source code program length is the number of Lines Of Code (LOC). Many different schemes have been proposed for counting LOC. In order to avoid any confusion that can easily be created if the precise definition of the LOC is not

provided, a simple and widely accepted way of counting LOC based on number of semicolons is used in this study.

Reuse is a product attribute partially related to the software size. Different entities can be reused in the software development process (design, code), and reuse can be implemented in many different ways (e.g. verbatim or slightly modified). Consequently, different aspects of reuse are measured using various software metrics. A useful set of metrics is the amount of reuse metrics used to assess and monitor reuse improvement and effort. In general, these metrics are defined as the ratio of the amount of the reused lifecycle item reused and the total size of that item. The form of this metric based on LOC would be ratio of LOC in the module and the total LOC in the module (Frakes and Terry, 1996).

The WebMetrics tool for software metrics collection (Section 5.3), used and improved in this study, also supports collection of reuse metrics.

For measurement of the amount of functionality in a system, function points are suggested by Albrecht (Albrecht and Gaffney, 1986). Function points are based on subjective judgment and are mainly used as a part of the effort estimation method.

Although related to the size, complexity represents another interesting product attribute that can be interpreted in different ways. For example, Fenton and Pfleeger (1997) refer to the computational complexity, algorithmic complexity, structural complexity, and cognitive complexity. Computational complexity reflects the complexity of the underlying problem. Algorithmic complexity measures the complexity of the algorithm used to solve the problem. Structural complexity is used to quantify the structure of the software that implements the algorithm. Cognitive complexity measures the effort required to understand the software implementation. Complexity of the object-oriented software design is discussed in the following section.

There are other useful internal product attributes besides size and complexity. Structural properties of software can help understand some difficulties in software development.

McCabe (1976) proposed that software complexity could be measured by the cyclomatic number of the program's flow graph. For a program with flow graph F, the cyclomatic number v measures the number of linearly independent paths through F and can be calculated as:

$$v(F) = e - n + 2$$

where F has e arcs and n nodes. Although the cyclomatic number cannot be used as a general complexity measure (Fenton and Pfleeger, 1997), sometimes it is a useful indicator of the maintenance effort.

3.3. Object-Oriented Metrics

Software can be produced in various ways. The traditional approach is to use a procedural language. However, there are other alternatives, such as object-oriented software development. The popularity of the object-oriented methods caused the need for appropriate object-oriented measures.

In the definition of the object-oriented systems by Yand and Weber (1990), the world is composed of substantial individuals that possess a finite set of properties. Collectively, an individual and its properties constitute an object. A class is a set of objects with common properties. Attributes such as coupling, cohesion, inheritance, and object complexity are defined for the classes of an object-oriented system.

As mentioned, relatively simple and well-understood set of CK metrics is used in this study. This suite of six metrics shows a good potential as a complete measurement framework in an object-oriented environment (Mendonça and Basili, 2000). Depth of inheritance tree (DIT) for a class corresponds to the maximum length from the root of the inheritance hierarchy to the node of the observed class. Another metrics related to inheritance is the number of children (NOC), representing the number of immediate descendants of the class in the inheritance tree. Coupling between objects (CBO) is defined as the number of other classes to which a class is coupled through method invocation or use of instance variables. Response for a class (RFC) is the cardinality of the set of all

internal methods and external methods directly invoked by them. The number of methods (NOM) is used as a simplified version of more general weighted methods count (WMC). The number of internal methods is extracted instead of forming a weighted sum of methods based on complexity. The lack of cohesion in methods (LCOM) is defined as the number of pairs of non-cohesive methods minus the count of cohesive method pairs, based on common instance variables used by the methods in a class. Since the analyzed code is written in C++, source lines of code are counted using semicolons.

A number of alternative object-oriented measures have been proposed. Some of them account for deficiencies of the CK suite (Li, 1998). The metric suite proposed by Li (1998) consists of the number of ancestor classes (NAC), number of local methods (NLM), class method complexity (CMC), number of descendent classes (NDC), coupling through abstract data type (CTA), and coupling through message passing (CTM). Marchesi (1998) introduces metrics for object oriented analysis models in UML. Nesi and Querci (1998) propose a set of complexity and size metrics for effort evaluation and prediction, providing also a validation for some of them. Reyes and Carver (1998) define an object-oriented inter-application reuse measure. Shih *et al.* (1998) propose a concepts of unit repeated inheritance and inheritance level technique for measuring the software complexity of an inheritance hierarchy. Bansiya and Davis (1999) introduce Average Method Complexity (AMC) and Class Design Entropy (CDE) that measure the complexity of a class using the information content. Kamiya *et al.* (1999) propose revised set of CK metrics for software with reused components. Miller *et al.* (1999) propose four new measures of hierarchy, inheritance, identity, polymorphism, and encapsulation in an object-oriented design. Teologlou (1999) describes the predictive object points for size and effort estimation.

3.4. Validity of Software Metrics

With a variety of the metrics proposed in software engineering, there is a clear need for validation whether a specific measure captures the attributes it claims to describe (Weyuker, 1988). Although it is essential to validate characteristics of a

measure, it is also important to determine whether the measure is part of a valid prediction system, i.e., to demonstrate that the measure is useful for estimation and prediction of some dependant variable in the software development process (Fenton and Pfleeger, 1997). This means that a measure must be viewed in the context in which it will be used; a measure may be valid for some uses but not for others. However, a measure can serve only one of these purposes, i.e., a measure should not be rejected as invalid just because it is not a predictor.

It is said that a measure is valid in the narrow sense (internally valid) if it is useful for assessment purposes. If the measure is valid in the narrow sense, it is also called internally or semantically valid. If a measure is also a component of a prediction system, it is said to be valid in the wide sense (Fenton and Pfleeger, 1997).

The best way to validate a software measure is to use it on multiple datasets and assess its usefulness for description of measured attributes or as a component of a prediction system. In this study, the CK metrics suite is used to capture defect behavior of the classes in the system.

3.5. The Goal Question Metric Paradigm

Measurement results can be used and interpreted in a variety of ways: for cost estimation, reliability purposes, maintainability etc. The purpose of measurement should be clearly stated. In addition, the customer, the manager, and the developer all view the product and the process from different perspectives. Thus, they are interested in different aspects of the project with different levels of detail.

It is essential to measure what is needed and useful for the organization, rather than what is convenient or easy to measure. Such software metrics programs often fail because the resulting data are not useful to the process.

These aspects of software measurement are summarized in the Goal Question Metric (GQM) paradigm (Basili and Rombach, 1988). This paradigm represents a mechanism for formalizing the characterization, planning, construction, analysis, and learning tasks in software engineering. It represents a systematic approach for

setting project goals customized for a specific organization and defining them in an operational way.

To clearly specify a set of operational goals, the measurement process is organized in a top-down order.

- The GQM approach provides such a framework involving three steps:
- List the major goals of an organization or a specific project.
- Derive a set of questions from the goals in order to determine if the goals are being met.
- Decide what must be measured in order to be able to answer the questions adequately.
- The use of the resulting data is also defined in this way, as will be discussed in Section 5.

Different sets of guidelines exist for defining product-related and process-related questions in the GQM framework (Basili and Rombach, 1988). Product related questions are formulated for the purpose of defining the product attributes, such as cost, changes, or defects. They define a specific quality perspective of interest, e.g. reliability. Process-related questions are formulated for the purpose of defining the a specific quality perspective of the process quality, such as reduction of defects or cost effectiveness.

The goal of this study is to demonstrate how the advanced software engineering models can be used for assurance and improvement of software quality. Two factors with great impact of the quality of software products are allocation of resources throughout the development process and the design of the system. External product quality metrics - historical data about occurrence of SRs over time - are used for building the models to support resource allocation. Internal product metrics - object-oriented design metrics from the CK metrics suite - are used to identify the critical defect prone classes in the system early in the development process. This information can be also used for efficient resource

allocation and for improvements in the design of the system. The GQM framework used in this study is presented in Table 1.

Goal	Question	Metrics
Improve quality of the software	How to effectively allocate available resources?	Data about occurrence of SRs over time with additional information about types of SRs
	How to recognize critical defect-prone, effort-consuming classes in the system early in the development process? How to improve design?	CK object-oriented design metrics suite

Table 1: GQM framework used in this study

What is not evident from the GQM tree or table is the model needed to combine the measurement in a sensible way so that the questions can be answered. The GQM approach must be supplemented by one or more models that express the relationships among the metrics. The appropriate models for software metrics data are discussed in Section 4, and the resulting framework for empirical investigation is presented in Section 5.

3.6. Types of Metrics Data

Measurement is by definition a mapping from an empirical relation system to a numerical system (Fenton and Pfleeger, 1997). The purpose of performing this mapping is manipulation of the data in numerical system, and drawing conclusion about attributes in the empirical domain. But not all measurement mappings are the same. In fact, the differences among the mappings can restrict the kinds of analyses.

The notion of measurement scale is essential for understanding these restrictions and application of appropriate analyses for the data.

Measurement scale is defined as the pair of the measurement mapping and the set of empirical and numerical systems.

A mapping from one acceptable measure to another is called an admissible transformation (Fenton and Pfleeger, 1997). The more restrictive the class of admissible transformations, the more sophisticated the measurement scale. For example, the class of admissible transformations for measuring length is very restrictive. All admissible transformations are of the form:

$$M'=aM$$

M is the original measure, M' is the new one, and a is a constant.

Scale	Admissible transformations	Simple examples	Software engineering	Relations	Statistics	Applicable methods
Nominal	1-1 mapping	Labeling	Types of defects	Equivalence	Mode, freq.	Non-parametric (Spearman correlation)
Ordinal	Any monotonically increasing function from M to M'	Preference	Severity of defects	Equivalence greater than	Median, percent	
Interval	Linear increasing function $M'=aM+b$ $(a>0)$	Temp. (°F, °C)	Timing of SRs occurrence	Equivalence greater than, ratio of intervals	Mean, variance	
Ratio	Linear increasing function passing through 0 $M'=aM$ $(a>0)$	Length	Program size	Equivalence, greater than, ratio of intervals, ratio of values	Geom. mean	Non-parametric and parametric (Pearson correlation)
Absolute	Identity $M'=M$	Counts	Defect count			

Table 2: Scale types (taken in part and with modifications from Fenton and Pfleeger, 1997)

Nominal scale is the most primitive form of measurement. There is no ordering among the classes defined by a nominal scale. Any distinct symbolic representation represents an acceptable measure on a nominal scale.

Classification of SRs according to their type (see Table 8) represents an example of a nominal scale. This measurement clearly provides information about additional attribute of SRs. In this way, a clear distinction between the different types of SRs is created, and every SR belongs to exactly one class. Any mapping that assigns different classes of SRs to different symbols represents an acceptable measure. Thus, the class of admissible transformations for a nominal scale measure is the set of all one-to-one mappings.

The ordinal scale contains information about the ordering of different categories, which does not have to be numeric. This ordering is based on the empirical attributes. Any mapping that preserves the ordering is an acceptable transformation. This scale contains only ranking information, so operations such as addition, subtraction, and other arithmetic operations are not defined.

Classification of SRs in terms of their severity or priority (see Table 7) defines an ordinal scale based on subjective assessment of SRs. There is a clear order relation "more severe than" between the different categories. To preserve this ordering, any monotonically increasing transformation represents an acceptable transformation.

The interval scale is more sophisticated and carries more information than nominal and ordinal scales. This scale captures information about the distance between the different categories. An interval scale preserves differences but not ratios. Addition and subtraction are acceptable, but not multiplication and division.

The timing of occurrence of SRs is an example of an interval scale. This timing can be measured in units of months, weeks, days, or hours, where each time is noted relative to a given fixed event - start of the project, for example. Consequently, the time of occurrence of an SR can be subtracted from the time when it was fixed. This operation is permitted for an interval scale and, in this

case, results in the time that was necessary to fix the problem. This measurement can also be transformed by setting a different starting date, e.g., new release – 300 days after start of the project, and using days instead of weeks. This would give:

$$M'=7M+300$$

Ratio scale is common in physical sciences and engineering. This measurement mapping preserves ordering, intervals between entities, and their ratios. There is a natural zero measure, representing lack of the measured attribute. All arithmetic operations can be meaningfully applied for this scale.

Time interval is a representative of this type of scale. Time necessary to fix an SR represents an example of a time interval. Size and length measures are also ratio measures. Consequently, software size expressed in LOC can be considered a ratio scale. Clearly, it is possible to have a software module with zero LOC. A ratio of the size of two programs can also be calculated.

Absolute is the most restrictive scale with respect to the admissible set of transformations. This measure represents counts of empirical entities. All arithmetic operations on such counts are possible.

Absolute scale is typical in software engineering. Number of SRs in a project and number of defects for a class are both examples of measures on an absolute scale. All the metrics from the CK suite are also absolute.

Understanding scale types is essential in determining what type of statistical analysis is applicable for data. For example, it is inappropriate to compute ratios with any scale bellow ratio scale. Table 2 presents a summary of the meaningful statistics for different scales types. This table is partially taken with modifications from Fenton and Pfleeger (1997). Every meaningful statistic for lower scale type is also meaningful for a higher one.

The logic behind the above proscriptions for use of statistical methods is that statistical measures should remain invariant under the admissible transformations for a particular scale (Briand *et al.*, 1996a). There are two types of invariance. First, invariance in value, where the numerical value of the statistic remains

unchanged under the admissible transformations. Second, invariance in reference, where the value of the statistic may change, but it would still refer to the same item or location. For example, the value of the median may change but it would still refer to the item at the middle of the distribution under monotonic increasing transformations. The item at the mean would remain the same under linear transformations even though the value of the mean changes.

3.7. Parametric and Nonparametric Statistics

The nature of the collected data determines the appropriate methods for their analysis. Most of the popular statistical techniques and tests require specific assumptions about the population of the sampled data. Furthermore, in many cases there are additional assumptions: that the distribution is roughly normal, that the variance are known or equal, or that the samples are independent (Freund and Simon, 1996). Oftentimes with software engineering data, it is doubtful whether all the necessary assumptions are met. For such cases, alternative nonparametric procedures based on less stringent assumptions are more appropriate.

Aside from the fact that nonparametric techniques can be used under more general conditions, they are often relatively easy to explain and understand. Nevertheless, the choice of a nonparametric statistic over an analogous parametric statistic, in general, results in loss of statistical power. This means that the probability of successful validating the hypothesis is reduced even if the hypothesis is valid (Briand *et al.*, 1996a). Having in mind that the statistical power is closely related to sample size, level of significance, and magnitude of the effect (e.g., correlation coefficient), it is even more difficult to prove a hypothesis in a software engineering environment where data are sparse, and correlations are typically low.

An example of an advanced non-parametric statistical technique is the gamma analysis introduced in Section 4.5 and used in Section 6 for analysis of patterns in occurrence of SRs.

Alternative to the parametric Pearson's correlation coefficient is nonparametric rank correlation, often called Spearman's coefficient. To calculate the rank

correlation coefficient for a given set of paired data, the data should be first ranked among themselves from low to high or from high to low. Then, the other variable is also ranked in the same way. The sum of the squares of the differences between the ranks is then found, and the correlation coefficient is calculated.

Having in mind that software metrics data are typically non-normally distributed, the Spearman's correlation is the general method applicable for measurement of magnitude of the effect of one variable to another. This method is used Section 7 for analysis of impact of object-oriented design on the defect behavior of the classes in the system.

3.8. Pragmatic Approach

There is a strong position of some software engineering practitioners that a pragmatic approach to software metrics analysis should be taken. In particular, Briand *at al.* (1996a) state that, in most cases, the measurement goals should be used to determine the type of analysis to be applied. Then, if a pattern is detected, the scientists should start analyzing the validity of the assumptions and considering alternative techniques. Furthermore, they demonstrate that some of the theoretical proscriptions would represent a substantial hindrance to the progress of empirical research in software engineering. Briand *et al.* (1996a) base their arguments on studies performed by statisticians and behavioral scientists over long period of time. Part of this earlier research demonstrated that observed scales sometimes fall somewhere between the ordinal and interval levels of measurement.

Although there is no doubt that some analyses could provide useful results even though all theoretical assumptions are not met in reality, it is very important to apply statistical method with assumptions closest to the empirical system. This is also important having in mind that inappropriate methods can give wrong and misleading results.

Briand *et al.* (1996a) confront findings by Mayer (1971) and Labovitz (1971) about treating ordinal and interval data. Mayer (1971) demonstrates that treating

ordinal data as interval can be inappropriate, leading to underestimation of relationships between the variables. In response to this, Labovitz (1971) states that Mayer's findings are applicable only to dichotomized scales. Such a scale clusters items on one part of the scale to the extent where they are almost overlapped, and stretches the other part of the scale.

Software engineering data is often dichotomized with most of the items concentrated in one part of the scale (typically low values), calling for extreme care in selection of appropriate statistical methods for analysis. Part of this study (Section 7) is dealing with the number of defects for classes. This dependent variable has high concentration of low and zero values. For this reason and with respect to the count type of the data, the appropriate statistical models, such as negative binomial model and its zero inflated extension, are investigated in detail in Section 4 and applied in Section 7.

4. Data Analysis in Software Engineering

Software engineering is based on questions about the process of software development and different sets of data collected from the process. The goal of software engineering is to understand the data and answer the questions in the best possible way. The experience and historical data in software engineering are best summarized in form of statistical models.

In general, a model is an abstraction of reality. It makes it possible to view the entity or concept from a particular perspective by removing less significant details. However, in order to provide useful results, it is essential to understand models' capabilities and limitations.

4.1. Types of Models in Software Engineering

Models are used for descriptive purposes, to explain relationships between the different components of the system. They can be formulated in different forms: as equations, mappings, diagrams etc.

In addition to their descriptive applications, models are also used for prediction purposes. Based on the baseline models developed on the historical data, it is

possible to make reasonable predictions of the future behavior of the analyzed system. An example of this use of models is demonstrated in the analysis of SRs in this study.

In order to solve a prediction problem, it is necessary to define a prediction system (Fenton and Pfleeger, 1997). A prediction system consists of a prediction model, an inference procedure, and a prediction procedure. The prediction model forms a specification of the system under consideration. The inference procedure defines how the parameters of the model are estimated. The prediction procedure is used to interpret the results of the model and the inference procedure in order to make the prediction.

Formally, a statistical model represents a set of assumptions about the joint distribution of the data (Lloyd, 1999). These assumptions can be divided into two components. First, the error distribution represents the assumption that the data comes from some specific family of distributions. Second, the systematic component of a statistical model represents the statement about the underlying pattern of the data.

The error distribution is used to describe the random variations of the data around the systematic component. For continuous data, for example, the normal distribution is often assumed (Lloyd, 1999). For discrete data, the most common distribution is the Poisson distribution. This issue is discussed in more details in Section 4.3, and appropriate regression models are applied for defect data in Section 7.

The systematic component of a statistical model commonly refers to the mean values. In this case, the systematic component is called the regression function of the model.

In order to build a sound statistical model of the underlying data, it is essential to correctly identify both components of the model. First, it is necessary to identify the error distribution of the data or give a quantitative statement how much is the distribution of the sample different from the assumed distribution. Definition of the regression function and estimation (fitting) of the unknown parameters of the

model represent the second step in building the model. Section 4.3 of this study provides more details on these two steps, and sections 6 and 7 employ different regression models for empirical data and provide the interpretation of the results.

4.2. Problems with Software Engineering Data

Software engineering datasets often have a number of characteristics that make analysis difficult (Gray and MacDonell, 1997). These difficulties include missing data, large number of variables (leading to lower degrees of freedom), strong colinearity between the variables, heteroscedasticity, complex non-linear relationships, outliers, and small size of datasets.

All these factors make the modeling process more difficult and the models of the process less reliable. Some of these problems can be at least partially overcome. For example, heteroscedasticity can be reduced by various transformations, and colinearity can be removed by principal component analysis (Briand and Wüst, 1999).

Another distinct area of concern is the acceptability and validation of the models. This includes the issue of the model explaining its predictions. Without sufficient semantic meaning attached to the model, a satisfactory level of validation is unlikely to be achieved. This problem is made even more serious by the small datasets commonly used for building the models, which sometimes produce counterintuitive results.

The final area of concern is generalizability. Since the first models based on software metrics were derived, attempts have been made to apply the models associated with them to other projects within the organization, or even to other organizations. Use of standard COCOMO coefficients in cost estimation is an example of such an attempt. The need to recalibrate a model for a new environment has been recognized and supported by numerous authors (Kemerer, 1987). Even the models that are easily regenerated, such as linear regression models, have problems with generalizability, given their susceptibility to outliers.

An awareness of possible approaches helps assure that the most appropriate model is developed through employing the most suitable alternative. In some cases, the combination of the methods may be useful, each providing estimates that can be combined. Such a case is described in this study when multiple SRGMs are used according to the different goals in the analysis (Section 6).

Small samples in software engineering make it difficult to identify the patterns that a given data set may have. Resampling schemas provide a way for dealing with this problem through generating artificial data sets from the original training set. A popular resampling technique is called the bootstrap method. There are many possible ways to generate bootstrap samples. In addition to bootstrapping, there are also other resampling schemes such as cross-validation and jackknifing.

Software engineering data often has to deal with large number of variables and low number of data points, resulting in low degree of freedom. To solve this problem, the stepwise regression method can be applied. In this regression method, all the available predictors are allowed to enter the model. Independent variables are selected to enter the model based on the p-value. The p-value is a measure of the statistical significance, representing the probability that the outcome of the analysis is just a result of chance. The lower the p-value is, the higher is the statistical significance of the result. The independent variable with the smallest p-value is entered at each step of the regression, if that value is sufficiently small. Variables already in the model are removed if their p-value becomes sufficiently large. The method terminates when no more variables are eligible for inclusion or removal. For example, $p = 0.01$ can be used as the criteria for a variable to enter the model, and $p = 0.05$ as the exit criteria.

It is common in software engineering, like in other fields, to have colinearity between measures capturing similar underlying phenomena. Briand and Wüst (2000) use the Principal Component Analysis (PCA) to determine the different dimensions captured by the design measures in their work.

Principal component analysis is a transformation typically associated with multidimensional data. PCA reduces the redundancy contained within the data by

creating a new series of variables (components). The mean of the original data is the origin of the transformed system with the mutually orthogonal transformed axes of each component. The resulting components are often more interpretable than the original variables.

Similar method for reduction of dimensions, called factor analysis, can also be used to group variables that measure mutually strongly related aspects into a single factor. Each of these factors represents a major dimension within the data (Gray and MacDonell, 1997).

The basic and most popular regression models, such as the linear regression model, assume homoscedasticity of the error distribution for the data (see Section 4.3.1). This means that the errors have a constant variance independent on the predicting variables. When the variance differs across observations, the errors are heteroscedastic. Possible way to deal with this problem is to transform the dependent variable in the analysis in order to improve homoscedasticity. However, this approach has to be carefully applied with consideration to the type of the dependent variable and the possible transformations for the corresponding measurement scale (discussed in Section 3.6).

For the models to be useful, it is necessary to identify their level of generalizability. Clearly, models developed for a specific company need not work for a different company. Furthermore, the same models might not be applicable within the same company for different teams or projects. In order to fully understand the characteristics of the model, it is necessary to clearly specify the environment for which the model was developed.

To validate generalizability of a model, it is useful to apply the model with necessary adjustments to different datasets. This could show that the modeling approach and the form of the model are successful for different environments, different projects, or even for different application domains. In this study, the models are developed using same methodology for fourteen different projects, developed in four companies, in two different application domains,.

Another problem with the software engineering data is that they are unbalanced. A balanced dataset should have equal number of available data points for each combination of the values of the independent variables. Clearly, such datasets are extremely rear in empirical data coming from the software industry. The two major problems caused by lack of balance are that the impact of factors can be concealed and that spurious correlations can be observed.

Kitchenham (1998) proposes a procedure for analyzing unbalanced datasets. This method is based on the forward pass residual analysis, similar to stepwise regression, to identify the most significant factors. The procedure is demonstrated on two simple artificial datasets with only three ordinal-scale independent variables with three levels each.

Author states that the procedure is easily extendable to ratio, interval, or absolute scale factors. However, the suggested way for dealing with some problems, such as non-normal distribution of the dependent variable, is to use non-linear transformation of the data. This approach has serious limitations depending on the scale of the measurement data used in the analysis (Table 2). Although potentially useful method for specific datasets, this approach is far from being general.

In software engineering, it is necessary to calculate probabilities for values that the analyzed random variable can assume. The probability distribution, in the form of a table, graph, or formula, provides this information.

Although the normal distribution is often assumed for the software engineering data, this approach is not always properly justified. This is particularly the case for variables of the count type that are common in software metrics. In Section 7 of this study, the dependent variable number of defects for a class (measured on the absolute scale) is modeled.

4.3. Regression Models

Treating count variables as continuous, although being common practice (Fenton and Neil, 1999; Gray and MacDonell, 1997), also endorsed by Briand *et al.* (1996), may result in inefficient and biased models (Long, 1997). Discreteness of

the dependent variable leads to conservative confidence intervals, i.e., overestimated significance level for dependent variables.

The most common distributions applied to count data are based on the Poisson and multinomial distributions (Lloyd, 1999). The Poisson distribution is particularly suitable for counting events occurring over time. In the corresponding PRM, the Poisson distribution determines the probability of a count, where the mean of the distribution is a function of the independent variables. PRM has been used in software engineering for modeling the number of faults (Graves *et al.*, 2000) and the effort expressed in hours (Briand and Wüst, 1999). PRM requires equidispersion, i.e., equality of the conditional variance and the conditional mean of the dependent variable. When conditions for the PRM are not met, e.g., in case of high conditional variance of the dependent variable, the Negative Binomial (NB) distribution and the associated NBRM can be used (Lloyd, 1999; Briand and Wüst, 1999).

It is common in software metrics data that the number of zeros exceeds the prediction of both PRM and NBRM. Zero-inflated count models explicitly model the number of predicted zeros (Lambert, 1990).

The following subsections explain these models.

4.3.1. Linear Regression Model

Linear regression model is very popular method in software metrics studies (Fray and MacDonell, 1997). It is used in many different ways, often in combination with various transformations to permit non-linearity.

The linear regression model can be written as:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n + \varepsilon$$

where y is the dependent variable, the x's are independent variables, and ε is a stochastic error. The β_1 through β_n are the parameters that indicate the effect of a given x on y. β_0 is the intercept that indicates the expected value of y when all of the x's are 0.

A number of assumptions are added to complete the specification of the model (Long, 1997). The first two assumptions concern the independent variables.

Linearity represents the assumption that the dependent variable is linearly related to the independent variables used in the model. Nonlinear relations in the model are possible through the inclusion of transformed variables.

Second assumption, colinearity, states that the independent variables x used in the model have to be linearly independent. This means that none of them is a linear combination of the remaining predictors in the model.

A second set of assumptions concerns the distribution of the error. Error ε can be thought of as an intrinsically random, unobservable influence on the dependent variable. Alternatively, ε can be viewed as the effect of a large number of variables excluded from the model that individually have small effect on the dependent variable.

The assumption of the zero conditional mean requires that the conditional expectation of the error is equal to zero. This means that, for a given set of values for the independent variables, the error is expected to be zero.

Furthermore, the errors are assumed to be homoscedastic and uncorrelated. Homoscedasticity represents the constant variance of error, independent on x's. The errors are also assumed to be uncorrelated across different observations.

Finally, it is assumed that errors, as a combination of many small unobserved factors, are normally distributed (Long, 1997).

It is often the case that most of the assumptions of the linear regression model are not satisfied for the empirical software engineering data. More appropriate models to deal with such datasets are discussed in the following subsections.

4.3.2. Ordinal Least Squares and Maximum Likelihood Methods

Although ordinal least squares (OLS) is the most frequently used method of estimation for regression models, its application for fitting the parameters of the model is justified if error distribution is assumed to be normal. On the other hand, the maximum likelihood (ML) method provides a general solution for fitting of

the model parameters for non-normally distributed data, when the underlying distribution is known or assumed. The OLS and the ML estimates of model parameters are approximately the same for the linear model if error distribution is assumed to be normal. ML is also applicable for models, such as PRM, where the variance of the data is not constant (Tryfos, 1998).

ML method is designed to maximize the probability that the model represents the best fit to the empirical data. This estimator is consistent, i.e., the probability that this estimator differs from the true parameter by an arbitrary small amount tends toward zero as the sample size grows. The variance of the ML estimator is the smallest possible among consistent estimators. This feature is usually referred to as asymptotic efficiency of the ML estimator. Thus, the ML estimators are used for the models in this analysis in Section 7.

4.3.3. Poisson Regression Model

The Poisson process is a simple model for occurrence of random variables that assumes the probability of an arrival in a small interval determined by the independent variables is determined only by the size of the interval, not on the history of the process to that time (Papoulis, 1991). A Poisson distribution is the distribution of the numbers of events resulting from a Poisson process.

The Poisson distribution for a dependent variable y, and a vector of n independent variables $\boldsymbol{x}=(x_1, \ldots x_n)$ is given by:

$$Pr(y|\boldsymbol{x}) = \frac{e^{-\mu} \cdot \mu^y}{y!}$$

where μ is the mean value of the dependent variable.

The Poisson distribution requires equidispersion of the data, that is, the conditional mean and the conditional variance of the dependent variable should be equal (Briand and Wüst, 1999; Lloyd, 1999):

$$E(y|\boldsymbol{x})=Var(y|\boldsymbol{x})$$

In practice, the conditional variance of the dependent variable in the model is often higher than its conditional mean, i.e., the dependent variable is

overdispersed. The main cause of the overdispersion is failure of the Poisson distribution to account for heterogeneity in the data. Overdispersion seriously compromises the goodness of fit of the model (Lloyd, 1999), resulting also in over-estimated statistical significance of the predictors in the model (Cameron and Trivedi, 1986).

The PRM accounts for heterogeneity in the data based on the observed characteristics of items, i.e., based on the independent variables. To fully define the PRM, a regression function describing the underlying pattern, i.e., the mean of the data has to be defined in combination with the error distribution. The goal of statistical analysis is to find a simple regression function that successfully models the main behavior of the data.

The exponential regression function, corresponding to the multiplicative model for the means, is commonly used with the Poisson distribution (Long, 1997; Lloyd, 1999). The conditional mean is given by:

$$\mu(y \mid \boldsymbol{x}) = e^{\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n} = e^{\boldsymbol{x}\boldsymbol{\beta}}$$

where $\boldsymbol{\beta}$ is the vector of model parameters.

4.3.4. Negative Binomial Regression Model

Empirical data are often over-dispersed, i.e., the value of the conditional variance is higher than the conditional mean of the dependent variable in the PRM. The main reason for this is the lack of complete control over experiments, attributing a great part of the variability in the observed data to unknown sources. This is also known as unexplained heterogeneity.

An extension of the PRM, the negative binomial regression model, allows the conditional variance of the dependent variable to exceed the conditional mean.

The NBRM can be derived from the Poisson distribution based on the unobserved heterogeneity by accounting for the combined effect of unobserved variables omitted from the original model (Gourieroux *et al.*, 1984). In the NBRM, the mean μ is replaced with the random variable $\tilde{\mu}$:

$$\tilde{\mu} = e^{x\beta+\varepsilon}$$

where ε represents a random error uncorrelated with $\boldsymbol{x}$.

The relationship between $\tilde{\mu}$ and the original μ is:

$$\tilde{\mu} = e^{x\beta}e^{\varepsilon} = \mu e^{\varepsilon} = \mu\delta$$

With assumption that $E(\varepsilon) = 0$, the expected count after adding the new source of variation is the same as it was for the PRM, i.e., $E(\tilde{\mu}) = \mu$.

For a given combination of independent variables in the NBRM, there is a distribution of μ's rather than a single value. Consequently, the probability distribution function for $\delta = e^{\varepsilon}$ must be specified to solve the probability for the dependent variable. The resulting distribution is a combination of the Poisson distribution and another probability distribution.

$$Pr(y \mid \boldsymbol{x}) = \int_0^{\infty} [Pr(y \mid \boldsymbol{x}, \delta) \cdot g(\delta)] d\delta$$

Gamma distribution (with positive parameter v) is commonly assumed for δ due to the closed form of the resulting distribution:

$$g(\delta) = \frac{v^{v}}{\Gamma(v)} \delta^{v-1} e^{-\delta v} \text{ for } v > 0$$

The resulting combined NB distribution is given with:

$$Pr(y_i \mid \boldsymbol{x}_i) = \frac{\Gamma(y_i + v)}{y! \cdot \Gamma(v)} \cdot \left(\frac{v}{v+\mu_i}\right)^{\mu} \cdot \left(\frac{\mu}{v+\mu_i}\right)^{y_i}$$

where v is a positive estimated parameter, and Γ stands for the Euler gamma function:

$$\Gamma(x) = \int_0^{\infty} t^{x-1} \cdot e^{-t} dt$$

For the NB distribution, the conditional mean of the dependent variable is the same as for the PRM while the conditional variance of the dependent variable is quadratic in the mean μ:

$$Var(y \mid x) = \mu\left(1 + \frac{\mu}{v}\right)$$

Since v is positive (Long, 1999) and μ for count variables is also positive, the variance exceeds the conditional mean of the Poisson distribution. The v^{-1} is usually referred to as the dispersion parameter α since increasing α increases the conditional variance of y. Consequently, a low value of α represents a low level of over-dispersion.

The NB distribution corrects three main sources of poor fit that are often found when the Poisson distribution is used.

First, the variance of the NB-distributed dependent variable exceeds the corresponding variance of the Poisson distribution for the given mean.

Second, the increased variance in the NB results in substantially larger probabilities for small counts.

Third, the probabilities for larger counts are slightly larger in the NB distribution.

All three aspects of the NB distribution in comparison with the Poisson distribution are presented in Figure 2. In the first part of the graph, probability of low values given by the NB distribution is higher than the one given by Poisson distribution. A similar behavior of the two distribution functions can be observed for the larger counts, where NB distribution also has higher values. For given mean for both distribution μ=3 and v=0.84 for NB distribution, the Var_{NB} = 13.69 is greater than $Var_{Poisson}$= 3.

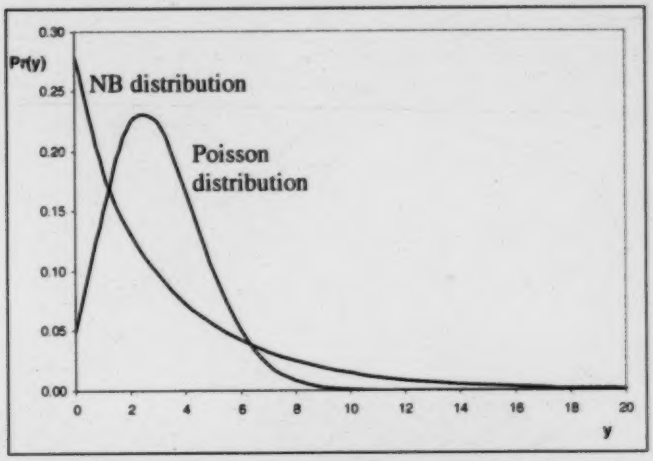


Figure 2: Poisson and NB distribution for μ=3

The resulting NBRM is the most commonly used model based on the combination of the Poisson distribution with other distributions (Long, 1999).

With increased α, the probability of zero values in the NB is increased. For sufficiently big α, the conditional mode for all the values of the dependent variable becomes equal to 0.

An alternative way to derive the NB distribution is based on the idea of the contagion process (Eggenberger and Pólya, 1923). The contagion is the process where analyzed items with a set of the independent variables initially have the same probability of certain event, but this probability changes as events occur over time. A process is contagious if the occurrence of events changes the future behavior of the process. Consequently, contagion violates the assumption of independence in the Poisson distribution. Both the unobserved heterogeneity and contagion can result in the NB distribution of the dependent variable. The heterogeneity is thus sometimes referred to as spurious contagion.

The NBRM model can be estimated by the maximum likelihood method, maximizing the likelihood equation:

$$L(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{X}) = \prod_{i=1}^{N} Pr(y_i \mid \boldsymbol{x}_i)$$

4.3.5. Zero Inflated Regression Model

The underprediction of zeroes in the PRM is partially resolved by the NBRM's increased conditional variance for the same conditional mean. On the other hand, zero-inflated models change the mean structure in order to explicitly model the occurrence of zero counts (Long, 1997). These models also increase the conditional variance of the dependent variable, as explained bellow.

Zero-inflated models allow the possibility that different processes generate zero counts and positive counts. They assume that two different groups form the population. An item belongs to one of the groups with probability ψ and to the other with probability $1-\psi$. This probability is determined from the characteristics of the item (Lambert, 1992; Greene, 1994). Items in the first group always have zero counts. Such items are different from those that have zero counts with a certain probability. These latter items belong to the second group together with the items with non-zero counts. The concept of two groups represents a discrete, unobserved heterogeneity since it is not known to which of the two groups an item with a zero count belongs.

In the group with items that are not always equal to zero, the resulting counts are governed by the PRM or the NBRM. For the NBRM, zeroes in this group occur with probability:

$$Pr(\, y=0 \,|\, \boldsymbol{x}\,) = \left(\frac{\nu}{\nu+\mu}\right)^{\mu}$$

where $\mu_i = e^{\boldsymbol{x}_i\boldsymbol{\beta}}$.

The overall probability of zeroes is a combination of the probabilities of zeroes from each group multiplied by the probability of an item belonging to a particular group.

The resulting model has the following form:

$$Pr(y=0|\boldsymbol{x}) = \psi(\boldsymbol{x}) + (1-\psi(\boldsymbol{x}))P_M(y=0|\boldsymbol{x})$$

$$Pr(y\neq 0|\boldsymbol{x}) = (1-\psi(\boldsymbol{x}))P_M(y|\boldsymbol{x})$$

where $P_M(y|\boldsymbol{x})$ is the probability given by the PRM or NBRM.

In the combined zero-inflated model, probability ψ is determined by either a probit or logit model $\psi=F(z\gamma)$, where F is the normal or the logistic cumulative distribution function, respectively (Long, 1997). In this study, the ZINB model based on logit probability for zero counts is used.

Variance of the ZINB model is given with (Long, 1999):

$$Var(y|x,z) = \mu(1- \psi)[1 + \mu (\psi + \alpha)]$$

The variance of the ZINB model exceeds variance of the PRM or any non-zero ψ.

The corresponding maximum likelihood method is available for zero-inflated models. The predictor vector z can (but does not have to) be the same as the vector x in the original PRM or NBRM. Clearly, if z and x are equal, i.e., if the same predictors are used in both parts of the model, the resulting model has twice as many parameters as the corresponding NRBM.

4.4. Software Reliability Growth Models

Various methods for the software reliability management and control operate on the data describing time between failures, failure rate, or cumulative count of failures over time (Littlewood, 1981). A Software Reliability Growth Model (SRGM) is a formal equation describing the cumulative number of errors discovered over time.

Based on the error-detection rate, SRGMs can be classified as concave or S-shaped. S-shaped models start with convex shape, reflecting the initial learning phase during which the detection rate increases (Lyu, 1996). Convex shape then gradually becomes concave as the time progresses. Both types of models assume finite number of errors in the software product. As most of the errors are detected and the product reaches the stable state, the error-detection rate decreases in both types of SRGMs.

Table 3 contains an overview of selected models. This table is an adaptation of Table A in (Wood, 1996), extended with a more S-shaped version of the Weibul model (W-S).

Model	Shape	Properties	Reference
GO S-shaped (GO-S)	S-shaped	$a(1-(1+bt)e^{-bt})$ $a\geq 0, b>0$	Yamada *et al.* (1983)
Goel-Okumoto (GO)	Concave	$a(1-e^{-bt})$ $a\geq 0, b>0$	Goel and Okumoto (1979)
Gompertz (G)	S-shaped	$a \cdot b^{c^t}$ $a\geq 0, 0\leq b\leq 1, 0<c<1$	Kececioglu (1991)
Hossain-Dahiya/GO (HD)	S-shaped	$a(1-e^{-bt})/(1+ce^{-bt})$ $a\geq 0, b>0, c>0$	Hossain and Dahiya (1993)
Logistic (L)	S-shaped	$a/(1+be^{-ct})$ $a\geq 0, b>0, c>0$	Musa *et al.* (1987)
Weibull (W)	S-shaped	$a(1-e^{-b \cdot t^c})$ $a\geq 0, b>0, c>0$	Musa *et al.* (1987)
Weibull *more* S-shaped (W-S)	S-shaped	$a(1-(1+b \cdot t^c) \cdot e^{-b \cdot t^c})$ $a\geq 0, b>0, c>0$	This study
Yamada Exponential (YE)	Concave	$a(1-e^{-b(1-e^{-c \cdot t})})$ $a\geq 0, b>0, c>0$	Yamada *et al.* (1986)
Yamada Raleigh (YR)	S-shaped	$a(1-e^{-b(1-e^{-c\frac{t^2}{2}})})$ $a\geq 0, b>0, c>0$	Yamada *et al.* (1986)

Table 3: Software reliability growth models (adopted version of Table A in Wood, 1996)

The time component in the models listed in Table 3, and in software reliability models in general, can be measured using calendar time, execution time, or the number of tests performed (Lyu, 1996). Since, in general, resources and effort in the development and testing process are changing over time, using effort information in combination with time would result in a realistic estimation. Effort alone is not capable of completely describing the servicing process since it does not account for the learning process, which is also dependent on time.

In this study, the calendar time information about occurrence of SRs is used. This type of information is easily available to project managers, making this approach widely applicable.

Since there is no single "best" general software reliability growth model, it is necessary to select one or more models that are most suitable for a particular software project and goal in the analysis (Littlewood, 1981).

A multimodel approach introduced in Section 6 of this study is on multiple criteria for effectiveness, predictive validity, goodness of fit, capability, and simplicity of the models.

4.5. Gamma Analysis

Gamma analysis is a non-parametric statistical technique used to identify the general order of different kinds of entities in a sequence. It is also used to quantify the amount of distinctness or overlapping of the time of their occurrences. Different entities being analyzed are usually referred to as "phases".

A nonparametric statistic called gamma score, which assesses the quantity of phases of one type in series that coming before or after the phases of another type, is computed using gamma analysis. The gamma score for a pair of phases is based on Gordan and Kruskal's gamma statistic (Pelz, 1985), defined with:

$$\frac{P-Q}{P+Q}$$

where P is she count of phases of one type preceding the other type of phase, and Q is the same count for the other phase with respect to the first phase. The result

of the analysis for a sequence of phases is a table of gamma scores for all the pairs of phases. The precedence score is defined as the average of gamma scores for that class, and represents the ranking of a phase in the total ordering of phases.

"Gamma maps" visualize the result of gamma analysis by ranking the phases based on the precedence scores and boxes are drawn to show the level of separation of the phases. Absolute value of precedence score is referred to as separation score, indicating the amount of distinctness of the phases. Phases with value of separation score greater than 0.50 are shown in separate boxes. Phases with separation scores between 0.50 and 0.25 are placed in continuous boxes. Phases with separation scores below the 0.25 value are shown in the same box.

Kemerer and Slaughter propose gamma statistics for the analysis of software evolution (Kemerer and Slaughter, 1999). It is found that Gamma analysis is also particularly suited to determine the orders of appearance of different kinds of SRs.

Gamma analysis is used to describe occurrence of SRs of different type over time. According to the usual practices in statistics (Pelz, 1985), the events occurring less than 4 times have been excluded from the analysis.

4.6. Methods and Criteria for Models Comparison

The goal of software engineering models is to create an accurate description of the software development process, to serve as a base for assessment and future improvements, and to predict future behavior of the process and the products.

It is necessary to select one or more models that are most suitable for the goals in the analysis and for particular software projects (Littlewood, 1981).

Various methods can be used to determine how well the model fits the data and how effective it is in prediction. These methods can be numerical or graphical, such as the correlation coefficients and the Alberg diagrams, explained in the following sections.

Applicability, effectiveness, and predictive ability are often used for assessment of models (Tian *et al.*, 1995). Applicability evaluates the performance of a model over time and across the different datasets, e.g., for different projects.

Effectiveness of the model quantifies its ability to be fitted to the actual observations. The ability of the model to predict future behavior in the software development process, based on the historical data, is usually referred to as predictive validity or predictive ability of the model (Fenton and Pfleeger, 1997).

Goodness of fit, based on the sum of square errors, is commonly used measure of effectiveness of the model (Yamada *et al.*, 1983). Other aspects of effectiveness can be evaluated using correlation coefficients and graphical methods, such as Alberg diagrams.

Capability and simplicity are also important factors for practical comparison of different models (Pressman, 1991). Model capability estimates usefulness of the information that the model provides for the software development process. Clearly, it is desirable to have a model with parameters that have some physical meaning, which can be well understood and interpreted. For a model to be really useful, it is essential that data collection process is time- and cost-effective. This characteristic of the model is referred to as the model simplicity. For example, the information about the modifications in the software is usually easily available to project managers from the company's software configuration management system, and this analysis is based on such dataset.

More detailed set of criteria is described in Section 6 where multi-model approach is used for description and prediction of occurrence of SRs over time.

4.6.1. Correlation Coefficient

Correlation coefficient is a measure of the extent to which two random variables track one another, i.e., magnitude of the effect of one variable to another. The parametric Pearson's correlation of two random variables x and y is given by:

$$r_{xy} = \frac{E[(x-\bar{x})(y-\bar{y})]}{\sigma_x \sigma_y}$$

where E is an expected value of the random variables, $\bar{x}$ and $\bar{y}$ are the expectations of x and y, and σ_x and σ_y are the standard deviations of x and y.

The correlation of two random variables can range from -1 to 1. The correlation coefficient is positive when the relationship between x and y is such that small values of y tend to go with small values of x, and large values of y tend to go with large values of x. When the link between the two variables is poor and knowledge of one of them does not help in prediction of the other one, the absolute value of the correlation coefficient has a low absolute value.

For typically non-normally distributed software metrics data, on different measurement scales, and non-linear relationships, the nonparametric Spearman's rank correlation is more general alternative to the parametric Pearson's correlation coefficient. To calculate the rank correlation coefficient for a given set of paired data, the data are ranked among themselves from low to high or from high to low. Then, the other variable is also ranked in the same way. The sum of the squares of the differences between the ranks is then found, and the correlation coefficient is calculated.

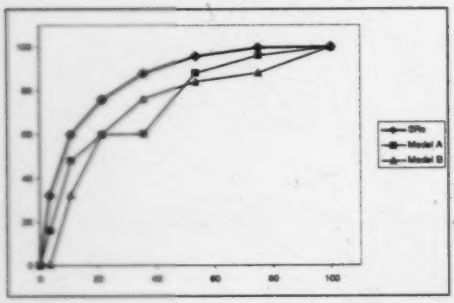
4.6.2. Alberg Diagrams

A graphical method called Alberg diagram, appropriate for scales below interval scale, can be used to compare performance of the different predictive models in terms of the criticality prediction for the classes in the system (Ohlsson and Alberg, 1996). This diagram is based on the Pareto principle (also referred to as 80/20 rule; Ebert and Baisch, 1998). Pareto rule states that a small number of modules (i.e., classes in Section 6 of this study) causes the major portion of the problems and, consequently, consumes the most effort in the system. Alberg diagram is formed by placing modules in decreasing order with respect to the number of defects. The x-axis is the percentage of the total number of modules, while y-axes represents the cumulative number of defects discovered in the corresponding classes.

By comparing the curves formed using the *observed data* and the results *estimated* by the model, the effectiveness of the model in identifying critical classes can be assessed. It is reported that models with lower correlation

coefficients can provide better prediction with respect to the criticality of the analyzed modules (Ohlsson and Alberg, 1996).

This can also be shown in the simple example in Figure 3. In this example, there are seven classes (C1 through C7) with corresponding SR counts. There are also two different models – A and B. Model A in this example has higher correlation with the empirical data (0.498) than model B (0.228). However, in the range from approximately 60% to 80% of the defects model B performs better in terms of criticality prediction.



SRs		Model A		Model B	
C1	80	C3	40	C7	1
C2	70	C1	80	C1	80
C3	40	C4	30	C2	70
C4	30	C7	1	C3	40
C5	20	C2	70	C5	20
C6	10	C5	20	C6	10
C7	1	C6	10	C4	30

Figure 3: An example of Alberg diagram

4.6.3. Sensitivity Analysis of the Models

Since the some software metrics, such as data about SRs are collected by developers, they are subject to human errors and imprecision. To assess sensitivity of the models to errors, a Monte Carlo simulation can be applied to determine what level of error affect the structure of the models and what can be tolerated without problems.

Monte Carlo Simulation is a computationally intensive mathematical technique for numerically solving differential problems. This technique has the advantage that it is a "brute force" technique that will solve many problems for which no other solutions exist. Monte Carlo simulation is appropriate when closed form or

other simple solutions does not exist for a problem. It is used extensively in highly complex problems.

In Section 6, white noise *N(0,s)* is added to the timing of the original data. The standard deviation, *s*, assumes values from half a day to an entire year. For each value of the standard deviation, the simulation is iterated 100 times. In each iteration, the associated models are extracted and their parameters are compared with the 95% confidence interval of the original models.

5. Empirical Investigation in Software Engineering

To answer the many questions of software engineering, it is necessary to provide an objective and scientific evidence. Three main ways for providing such evidence are surveys, case studies, and formal experiments (Fenton and Pfleeger, 1997).

5.1. Basic Research Techniques

A survey represents a retrospective study of a situation with the goal to document relationships within the system and outcomes of the changes made to the system. A survey is always done after en event has occurred. Software engineering surveys typically poll a set of data from software development process to determine the impact of a particular method, tool, or technique, or to determine trends or relationships.

In a survey, there is no control over the underlying process or system when a survey is performed. Because a survey is a retrospective study, the situation can be recorded and compared with similar ones, but it is not possible to manipulate variables.

Different levels of manipulation of the relevant variables are possible only in case studies and experiments. Case studies and formal experiments are usually not retrospective. The goals and plan of the investigation are made in advance.

A case study is a research technique for identification and documentation of the key factors that affect the system under analysis. Case studies usually look at a

typical project, and surveys are used to poll the behavior over large groups of projects.

A formal experiment is a rigorous controlled investigation of a system, where the factors of interest are identified and manipulated to investigate their effect on the outcome. Since formal experiments require a great deal of control, they tend to be relatively small, involving relatively small numbers of people and events.

To form a framework for an empirical investigation, it is necessary to form definition, plan, operations, and interpretation of the experiment (Basili et al., 1986). Definition of the experiment is used to set up a clear motivation of the experiment and to provide details about the object, purpose perspective, domain, and scope of the experiment. This information could be then used to improve the existing process and assure high quality of the products.

Objects of the experiment in software engineering typically are the development process and the resulting products. The domain and scope of the experiment should also be precisely defined, stating the characteristics of the analyzed project and the environment.

With a precise definition of the empirical study in place, it is possible to proceed to the planning phase. Planning covers issues of experiment design, criteria for comparison between the testing groups, and methods for measurement. Experimental design is based on the information about the size and other characteristics of the development teams, and specification of the projects that the groups will work on.

For preparation for operation of the empirical study or experiment, the developers and managers should be provided with basic training in the concepts being introduced. Data collection is then performed through the development of the projects. Finally, the quantitative and qualitative analysis is performed.

The results of the empirical investigation are then analyzed using various statistical methods applicable for the particular data sets. Criteria for assessment of the results of new practices are usually based on their effects on the quality of the developed software and the costs of the development.

To interpret the results of the experiment the context has to be clearly stated, and the impact of the treatment has to be described.

Although formal experiments theoretically have best potential to provide the most useful results, it is often very difficult to perform them. The primary reason for this is the restrictively high price of a formal experiment in an industrial environment. Even when a formal experiment is feasible from the cost perspective, for example in an academic environment (Williams, 2000), it might not be possible to replicate all the settings and control all the variables that might influence the outcome. This is particularly the case having in mind that software development is heavily dependent on the human factor.

Consequently, surveys and case studies represent the typical research techniques in software engineering. This study provides results of multiple case studies from different industrial environments.

5.2. Framework for Empirical Investigation

As discussed in Section 3.5, the GQM paradigm (Basili and Rombach, 1988) represents a systematic approach for setting project goals customized for a specific organization and defining them in an operational way. The measurement process is organized in a top-down order in three steps, starting from the major goals of the organization. A set of questions is derived from the goals. These questions determine the set of metrics to be used.

The questions in the GQM framework also directly influence the way in which the empirical investigation is performed. The definition of the experiment or other research technique described in Section 5.1, is directly related with the goal from the GQM paradigm (Basili and Selby, 1991). The design of the investigation is closely related to the components of the GQM paradigm where the set of questions is defined. The data collection process is directly defined by the selected set of metrics in GQM.

As discussed in Section 3.6, every software metric has an associated measurement scale and corresponding set of admissible transformations. The notion of

measurement scale is essential for application of appropriate data analyses techniques and models (Section 4).

The data collection process and type and quality of the data also directly influence applicable statistical methods (Kitchenham, 1998). Finally, the interpretation of the results depends on the initial set of goals from GQM, and the results of statistical analysis.

The resulting framework for empirical investigation in software engineering is graphically presented in Figure 4, using *has* and *defines* relationships between the entities in the framework.

The set of goals in this study, stated in the GQM form in Section 3.5, is based on improving the quality of the products. Set of metrics used in the study is based on two factors with great impact of software quality: allocation of resources in the development process, and the design of the system. Software metrics describing these two aspects of software development are collected in industrial environment. Particular care has been taken in investigation and proper allocation of statistical methods and model applicable for the collected metrics. The analysis of the collected data and interpretation of the results with respect to the initial goals is performed with in Sections 6 and 7.

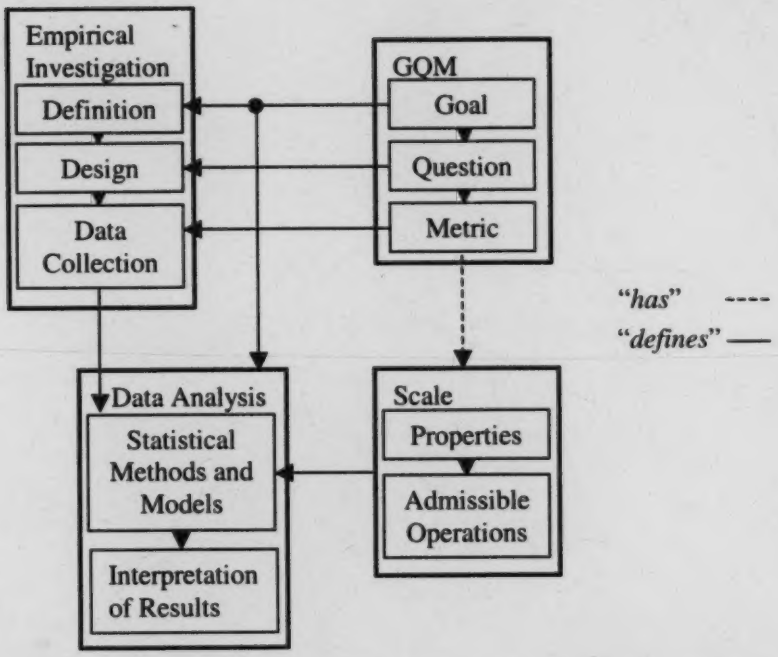


Figure 4: Framework for empirical investigation in software engineering

5.3. Data Collection Process

Tool used for data collection in this study is WebMetrics, a research system for software metrics collection (Succi *et al*, 1998). The main feature of the system is the variety of metrics extraction tools available for different languages – C, C++, Java, Smalltalk, and Rational Rose petal format. The WebMetrics tool is written in Java.

The metrics tool calculates a predetermined set of metrics shown in Table 4. New metrics can be programmed using the provided API.

Object-Oriented Metrics	Procedural Metrics	Reuse Metrics
Class LOC NOM DIT NOC	Function LOC McCabe's cyclomatic complexity Halstead volume	Internal and external reuse level, frequency, and density for classes, objects, and files

Object-Oriented Metrics	Procedural Metrics	Reuse Metrics
CBO RFC LCOM	Information flow (Fan-In/Fan-Out)	objects, and files

Table 4: Metrics collected by WebMetrics

5.3.1. Metrics Extraction

The field of software metrics is constantly changing. There is no standard set of metrics, and new measures are always being proposed. Metrics researchers have to modify their existing parser tools in order to accommodate the new measures. This is a real challenge since such tools usually have very complex parser-generator and language-semantics related source code. It is also easy for metrics researchers to inject errors while modifying the large amounts of code involved. Therefore, it would be desirable to decouple the information extraction process from the use of the information. More specifically, the language parsing should be decoupled from the metrics analysis portion of the process. This requires an additional layer of abstraction with an associated intermediate representation, as shown in Figure 5.

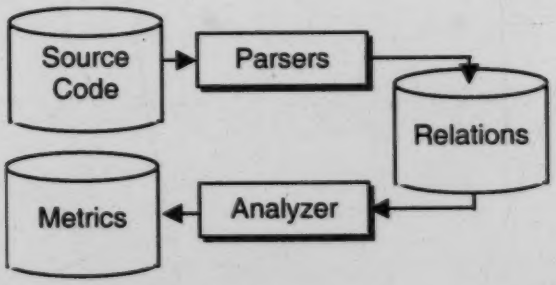


Figure 5: Relations as additional abstraction layer in software metrics analysis

The tool parses source files into relation files. The relations describe the existence and relations between entities found in the source files. Then, the tool analyzes all the relations and calculates metrics from them.

The relations produced by the metrics tools conform to the WebMetrics Relations Language (WRL) – a high-level, metrics-oriented intermediate representation used to convey the structure of a source program. The structure of a system is based on entities (such as classes and functions) and their interactions with each other. WRL describes a set of such relations. All of the WebMetrics language parsers output WRL.

Each metrics tool consists of a grammar parser, a symbol table, and supporting classes. The grammar parser recognizes the syntax of a particular language and is written in JavaCC (Metamata, 2001). The metrics tool is written in Java 2, and it can work on all platforms with adequate virtual machine.

The extra layer of abstraction inserted into the metrics analysis process creates a more modular architecture overall. There are tradeoffs involving this approach, but the benefits seem to outweigh the shortcomings.

Metrics researchers only need to deal with the high-level, metrics-oriented intermediate representation when adding or modifying metrics to calculate. This spares them from having to know intimidating details about how a language is parsed. What this means in the end is that modifications can be done more easily, more quickly, and with less chance of injecting errors into the existing source code.

In addition, the breakdown of the metrics extraction process into modules offers more opportunities for reuse. Each module and abstraction layer is a point of reference for reuse by other modules.

On the other hand, adding an extra layer of abstraction means that initial development time will be longer, since the developer needs to spend more effort in the design of the modules and the intermediate representation. However, the savings in maintenance effort later on in the development lifecycle offset this disadvantage.

Performance will likely degrade with the extra layer, but that is expected for having more flexibility.

Also, the intermediate representation needs to be designed very carefully. It has to be adequate enough such that all desired metrics can be calculated from that representation alone.

5.3.2. WebMetrics Relations Language

The relations are designed in form of a logic language, as Prolog-like clauses. This structure is ideal for describing language-entity relations.

Currently, the following relations are defined. They have been chosen to specifically facilitate the calculation of certain OO design and procedural metrics:

Relation	Description	Simple Example
hasLOC(entity, *x*)	The specified entity has *x* lines of code.	hasLOC(Stack, 6)
hasClass(entity, class)	The specified entity contains the specified (inner) class.	hasClass(Stack, Stack::Iterator)
hasMethod(entity, method)	The specified entity has the specified method.	hasMethod(Stack, Stack::push)
hasAttribute(entity, attribute, typename)	The specified entity has an attribute of the specified type.	hasAttribute(Stack, Stack::size, int)
hasMetric(entity, metric, value)	The specified entity has the specified value for a particular metric.	hasMetric(A.aMethod#0#, FanIn, 0).
hasFile(filename).	A parsed entity includes the specified file.	hasFile(D:/Parsers/Include/VC/include/winver.h).
extends(entity, class)	The specified entity is a specialization of the	extends(Stack, Collection)

Relation	Description	Simple Example
	specified class.	
calls(entity, method, *x*)	The specified entity called the specified method *x* times.	calls(Stack::push, Stack::isFull, 1)
usesAttribute(entity, attribute, *x*)	The specified entity uses the specified attribute *x* times.	usesAttribute(Stack::isFull, Stack::size, 2)

Table 5: Set of WebMetrics relations

5.3.3. The CK Metrics Example

As an example, the following table illustrates how the CK metrics can be expressed in terms of the relations, using a simplified set-based notation called SL (Succi and Uhlik, 1997).

Metrics expressed in terms of relations
WMC $wmc(X) = \lvert\{I : hasMethod(X, I)\}\rvert.$
DIT $parents(X) = \{I : extends(X, I)\}.$ $dit(X) = if\ (\lvert parents(X)\rvert \neq 0)\ \max(\{dit(I) : I\ in\ parents(X)\})\ else\ 0.$
NOC $noc(X) = \lvert\{I : extends(I, X)\}\rvert.$
CBO $methods(X) = \{I : hasMethod(X, I)\}.$ $attributedClasses(X) = \{I : hasAttribute(X, _, I)\}\ union\ \{I : Y\ in\ method(X);\ hasAttribute(Y, _, I)\}.$ $usedClasses(X) = \{I : usesAttribute(X, I, _)\}\ union\ \{I : Y\ in\ methods(X);\ usesAttribute(Y, I, _)\}$ $calledMethods(X) = \{I : Y\ in\ methods(X);\ calls(Y, I, _)\}.$ $methodClasses(X) = \{I : Y\ in\ calledMethods(X);\ hasMethod(I, Y)\}.$ $cbo(X) = \lvert attributedClasses(X)\ union\ methodClases(X)\ union\ usedClasses(X)\rvert.$
RFC $localMethods(X) = \{I : hasMethod(X, I)\}.$ $calledMethods(X) = \{I : Y\ in\ localMethods(X);\ calls(Y, I, _)\}.$ $rfc(X) = \lvert localMethods(X)\ union\ calledMethods(X)\rvert.$
LCOM $methods(X) = \{I : hasMethod(X, I)\}.$ $commonAttributes(X, Y) = \{I : usesAttribute(X, I, _), usesAttribute(Y, I, _)\}.$ $setQ(X) = \{(I, J) : I\ in\ methods(X), J\ in\ methods(X); I \neq J, \lvert commonAttributes(I, J)\rvert \neq 0\}$ $setP(X) = \{(I, J) : I\ in\ methods(X), J\ in\ methods(X); I \neq J, \lvert commonAttributes(I, J)\rvert = 0\}$ $diff(X) = (\lvert setP(X)\rvert - \lvert setQ(X)\rvert)/2.$ $lcom(X) = if\ (diff(X) \geq 0)\ diff(X)\ else\ 0.$

Table 6: Expression of CK metrics in terms of relations

This shows that the relations can be used to formally express the CK metrics. The metric values can then be calculated directly using these expressions.

6. Analysis of Service Requests

6.1. Discussion of the Experimental Data from Real-time Domain

This section focuses on four projects coming from a major North-American telecommunication company (Project A, Project B, Project C, and Project D). The original data has been transformed in an appropriate way to avoid revealing any confidential information, but still without any loss of accuracy or information.

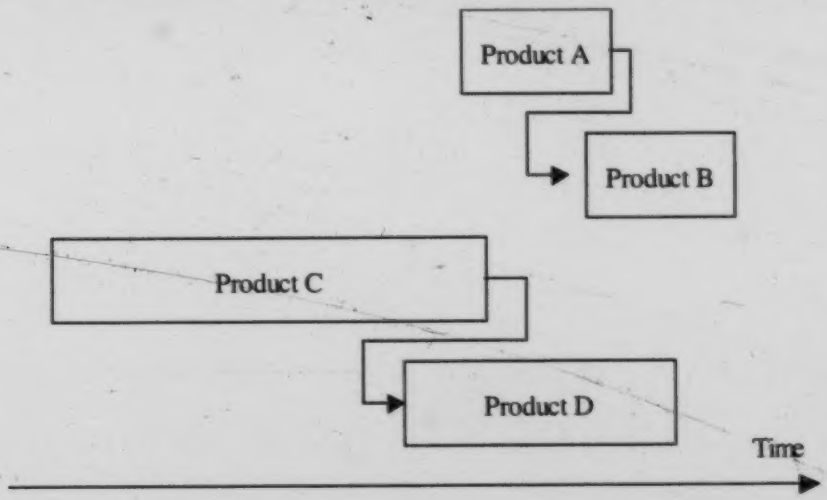


Figure 6: Temporal evolution of the four projects

Figure 6 contains the temporal evolution and the dependencies of the four projects. Projects A and C were the basis for the development of projects B and D. The development and testing of products lasted about one year for projects A and B, four years for project C and two years for project D. Projects A and B were the result of an acquisition of an external company and had a remarkable time pressure. Projects C and D come from the "regular" line of business. All the four projects are in the real-time telecommunication domain and their code was written mainly in C++.

The developers of the four projects had about the same education and skills – a BSc either in Electrical and Computer Engineering or in Computer Science. No particular pattern was observed in the assignment of developers to projects.

In total, there were 1295 different SRs for all the four projects (153 for Project A, 345 for Project B, 625 for Project C, and 172 for Project D).

The information on SRs is stored in a corporate repository. For each SR there is the time such SR was entered and how long it took to serve it. Figure 7 contains the arrival time of the SRs. The specific dates of arrival are omitted from the x axis for confidentiality.

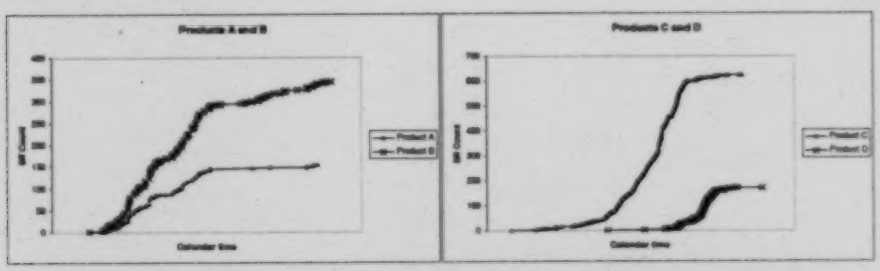


Figure 7: Arrival time of Service Requests for the four projects

Each service requests has an associated severity, which is subjectively determined by the tester following corporate guidelines. The company adopts an ordinal scale to quantify the severity, ranging from S1 – the most severe SR, to S5 – a cosmetic SR.

Table 7 contains the description of the severity levels used at the company and the breakdown of the SRs by severity for each project. There are very few SRs of level S5. This is considered a consequence of the target real-time application domain, where cosmetic issues are often not addressed.

Level	Description	A	B	C	D
S1	*Critical* – A problem that makes the whole system, a subsystem, or major feature/function non-operational. Issue is worked 24 hours a day, seven days a week,	5 (3.27%)	13 (3.77%)	74 (11.84%)	20 (11.63%)

Level	Description	A	B	C	D
	with the objective to solve it in 48 hours.				
S2	*Major* – Non-emergency failure of system operations with no significant effect to overall system performance. The objective is to start serving the request in 72 hours and to resolve the problem in one month.	51 (33.33%)	113 (32.75%)	267 (42.72%)	94 (54.65%)
S3	*Minor* – Item that can be worked around efficiently and does not cause serious problems. The objective is to solve the issue in 180 days.	78 (50.98%)	196 (56.81%)	250 (40.00%)	50 (29.07%)
S4	*Marginal* – Issue with no visible impact.	17 (11.11%)	21 (6.09%)	31 (4.96%)	6 (3.49%)
S5	*Ignorable* – Documentation and lowest impact requests.	2 (1.31%)	2 (0.58%)	3 (0.48%)	2 (1.16%)
Total		153 (100%)	345 (100%)	625 (100%)	172 (100%)

Table 7: Severity levels of the SRs in the four projects

SRs for projects A and B also have associated information on the type of the service request. Table 8 contains the description of such types and their frequencies.

Name	Description	A	B
Assistance (A)	Request for additional help with configuration of the system, or interpretation of the specification.	0 (0.00%)	12 (3.83%)
Deficiency (D)	Lack of support for the specified functionality.	28 (15.56%)	69 (22.04%)
Enhancement (E)	Request for an enhancement in functionality of the software.	6 (3.33%)	8 (2.56%)
Feature (F)	Possible request for a new feature or change of an existing one.	2 (1.11%)	11 (3.51%)
Hardware (H)	Problem traceable to hardware components of the system.	1 (0.56%)	2 (0.64%)
Information (I)	Request for an update of missing or incorrect information.	2 (1.11%)	5 (1.60%)
Other (O)	Other unclassified service requests.	2 (1.11%)	2 (0.64%)
Product service request (P)	Report of a system-level product defect.	47 (26.11%)	69 (22.04%)
Software (S)	Problem traceable to software.	92 (51.11%)	147 (46.96%)

Table 8: Types of SRs in projects A and B

6.1.1. *Extraction and Analysis of the Models for Timing of SRs*

In this section, the performance of the models describing the timing of SRs listed in Table 3 is assessed. The models are first analyzed across the four entire datasets. A separate analysis is then performed for the different severity levels and, for Project A, for the different components, to determine the impact and the use of such additional information, if any. The analysis on the types of SRs is not performed, since there are multiple types with very few numbers of SRs.

The parameters of the models are estimated using least square error regression on the available SR data. The statistical tool used to tune the model parameters employs an iterative estimation algorithm for finding the global minimum of the cost function. The bootstrap method is used to determine the confidence intervals for parameters of models.

As mentioned, the goal of modeling the occurrences of SRs is twofold:

To create an accurate description for assessment, comparison and improvement,

To predict the occurrence of the SRs early in the evolution of the SR serving process.

The first goal can be further organized in terms of goodness of fit, the accuracy of the final point, relative precision of fit, coverage of fit, and predictive ability of the model.

The *goodness of fit* represents how well the model fits the data, and therefore it is a reliable descriptor of the overall process, to be used for comparison and assessment. The goodness of fit is measured using the sum square errors.

The *accuracy of the final point* represents whether the model is able to determine the total final number of SRs. It is measured with E:

$$E = 100 \cdot \left| \frac{A - \alpha}{A} \right|$$

where A and α are respectively the true and estimated value of the total SRs served.

The *relative precision of fit* is the size of the bootstrap 95% confidence interval computed over the parameters of the model and normalized over the size of the interval of time of SRs arrival.

The *coverage of fit* is the degree to which the 95% confidence interval captures the oncoming service requests (shown in the *Data Coverage* column of Table 9).

Relative precision of fit and coverage of fit measure two complementary aspects of the fit that must be considered together to evaluate the value of a model: a very large 95% confidence interval might be able to capture most of the data, but it would be totally useless.

The *predictive ability* represents how early in the development the model is able to predict the final count of SRs. A model is considered to successfully predict the total count of SRs if the count is estimated within the ±10% range of the final number of SRs recorded. The predictive ability of a model is measured by computing the ratio of when the model successfully predicts the final number of SRs and the overall length of the interval of time of SRs arrivals.

Table 9 presents the results of the analysis and Table 10 provides the ranking of the models for each criteria.

The suitability of the models is highly variable, there is not a model performing the best in all cases. However, for the family of datasets used in this analysis, the choice can be restricted a few models that perform much better than the others.

For both projects A and B the group of models that perform reasonably well consists of G, GO-S, HD, L, W, and W-S. For projects C and D this group is reduced to HD, G, and L. (For an overview of SRGMs, see Table 3 in Section 4.4.) Only those three models were able to model the long initial concave shape of the SR occurrence.

From this analysis, it appears that a reliable estimation can be performed only using multiple models together.

	Goodness of Fit				Accuracy of Final Point				Predictive Ability			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
G	18	37	646	205	0.11	2.84	60.42	11.22	0.64	0.50	N/A	N/A
GO	103	207	7386	878	42.56	40.45	1294.11	312.58	N/A	N/A	N/A	N/A
GO-S	30	48	2563	463	8.01	2.68	3869.67	2803.78	1.00	0.75	N/A	N/A
HD	20	43	167	39	0.74	4.25	9.48	0.68	0.73	0.50	0.98	0.77
L	26	62	166	39	2.25	5.69	9.47	0.72	N/A	0.50	0.98	0.77
W	16	36	3104	366	0.98	4.08	132.69	14.21	0.18	0.17	N/A	N/A
W-S	16	33	1455	289	0.06	2.50	258.37	15.69	1.00	0.67	N/A	N/A
YE	104	212	10329	897	45.15	42.34	8.12	59.18	N/A	0.67	0.05	N/A
YR	17	43	3721	635	1.96	6.81	41.80	1.75	N/A	N/A	N/A	N/A

	Relative Precision of Fit				Coverage of Fit			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
G	440	159	52797	9217	89.71	5.88	95.27	99.01
GO	1509	1628	4950	1678	76.47	99.16	37.84	47.52
GO-S	613	107	4524	314	75.00	2.52	58.45	7.92
HD	306	977	908	234	42.65	40.34	55.41	51.49
L	457	417	736	695	80.88	41.18	47.30	73.27
W	1177	3467	17715	12368	89.71	73.95	99.32	99.01
W-S	1714	1444	68379	13280	98.53	99.16	99.66	99.01
YE	3882	4184	79115	1558	89.71	99.16	99.66	42.57
YR	6847	7007	2154	240	98.53	87.39	18.92	4.95

Table 9: Result of the analysis of the SRGMs

Rank	Goodness of Fit				Accuracy of Final Point				Relative Precision of Fit			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
1	W-S	W-S	L	HD	W-S	W-S	YE	HD	HD	GO-S	L	HD
2	W	W	HD	L	G	GO-S	L	L	G	G	HD	YR
3	YR	G	G	G	HD	G	HD	YR	L	L	YR	GO-S
4	G	HD	W-S	W-S	W	W	YR	G	GO-S	HD	GO-S	L
5	HD	YR	GO-S	W	YR	HD	G	W	W	W-S	GO	YE

	Goodness of Fit				Accuracy of Final Point				Relative Precision of Fit			
6	L	GO-S	W	GO-S	L	L	W	W-S	GO	GO	W	GO
7	GO-S	L	YR	YR	GO-S	YR	W-S	YE	W-S	W	G	G
8	GO	GO	GO	GO	GO	GO	GO	GO	YE	YE	W-S	W
9	YE	YE	YE	YE	YE	YE	GO-S	GO-S	YR	YR	YE	W-S

	Coverage of Fit				Predictive Ability			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
1	YR	GO	W-S	G	W	W	YE	HD, L
2	W-S	YE	YE	W	G	L, HD, G	HD, L	
3	G	W-S	W	W-S	HD			G, GO, GO-S, W, W-S, YE, YR
4	W	YR	G	L	GO-S, W-S		G, GO, GO-S, W, W-S, YR	
5	YE	W	GO-S	HD		YE, W-S		
6	L	L	HD	GO	GO, YE, YR, L			
7	GO	HD	L	YE		GO-S		
8	GO-S	G	GO	GO-S		YR, GO		
9	HD	GO-S	YR	YR				

Table 10: Ranking of the SRGMs

Building models using only SRs of a given severity and, for project A, only SRs coming from individual components does not appear to alter significantly in most of the cases the SRGMs.

Using the 95% confidence interval, no remarkable variation of the models across the different severity levels is detected. There is some possible indication that SRs of severity S1 may result in different models. However, such indication is not consistent across all the projects: 4 models out of the best 6 are different for project A, 2 out of 6 for project B, 1 out of 3 for project C and none for project D.

It could be therefore considered an artifactual result of the fewer data points of such level of severity.

The analysis performed on the individual components in project A does not reveal any difference.

Altogether, it appears that the information on severity and the specific components does not increase the performances of the SGRM for the projects under consideration.

6.1.2. Time to Resolve the SRs

As mentioned, it is important for managers to predict the time required to serve SRs. It is intuitive to expect a learning process and the decrease of the time to serve as the SRs servicing process progresses. The analysis of variance confirms this intuition in all the datasets and evidences that the most influential significant factor explaining the variation in time to serve is the calendar time (Figure 8).

The correlation coefficients between calendar time and time to serve are high for all the projects ($r_{\text{Project A}} = 0.82$, $r_{\text{Project B}} = 0.77$, $r_{\text{Project C}} = 0.67$, $r_{\text{Project D}} = 0.82$) with high level of significance (always, $p < 10^{-3}$).

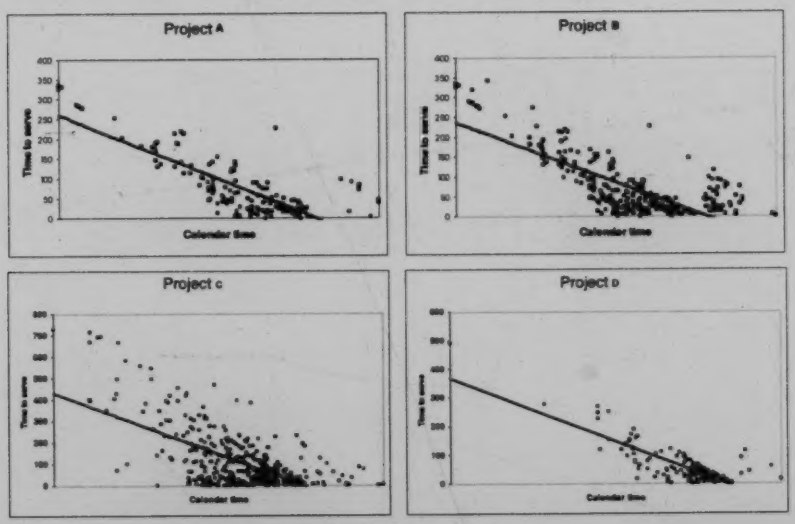


Figure 8: Scatterplot of the variation of the time to serve with the regression line

The linear regression coefficients are listed in Table 11.

	Slope	Sig.	Intercept	Sig.	r
Project A	-0.63	$<10^{-3}$	258.95	$<10^{-3}$	0.82
Project B	-0.53	$<10^{-3}$	235.89	$<10^{-3}$	0.77
Project C	-0.47	$<10^{-3}$	428.78	$<10^{-3}$	0.67
Project D	-0.65	$<10^{-3}$	365.24	$<10^{-3}$	0.82

Table 11: Coefficients of linear regressions

The high level of significance of the coefficients suggests the presence of linear relations.

Unfortunately, the linear models provide reasonable but not exceptional descriptions of the data. The distributions of residuals follow quite closely a normal distribution. However, the scatterplots of residuals evidence a variation of the spread of the data with calendar time, especially for projects C and D, and the Durbin Watson coefficients reveal for projects A and B the presence of positive autocorrelation.

Polynomial and logistic models have also been tried but without any improvement of the representations.

The effect of severity on servicing time are analyzed in the same way as for the SRGMs. Models specific to the single levels of severity are extracted and the 95% confidence intervals of the parameters of the models are compared.

Like for the case of SRGMs, the limited size of some of the datasets prevents from making general conclusions. It appears, though, that SRs of severity S1 results in models significantly different from other SRs in 3 out of 4 cases. The managers of the company explain this with the fact that such SRs are the only that require 24 hours /day, 7 days/week servicing. No remarkable differences were identified for the other levels. The difference of severity S1 may also be the cause of the patterns in the scatterplots of residuals that were previously observed. The models specific to each severity level have been analyzed; however, the limited

amount of data for certain levels of severity does not support any final conclusion. More careful investigation is clearly required.

For project A, no difference has been noted on the servicing time across components.

6.1.3. Kinds of SRs

Knowing the temporal distribution of the kinds of SRs over time allows a proper allocation of the right kinds of resources and enables an analysis of the overall evolution of the servicing process.

In this research, gamma analysis is applied to the severity of SRs (Figure 9) and, for projects A and B only, to the types of SRs (Figure 10). Not all severity levels and types of SRs appear in the gamma maps, since elements occurring less than 4 times cannot be included (Pelz, 1985).

With respect to the severity, the gamma analysis reveals that in general SRs with the lower priority tend to come first, while higher priority SRs occurred in the later stages of development. Such results have been reported to the managers of the company. They relate such results to the pressure that occurs at the end of the development of the projects, when, due to time constraints, the focus is on higher priority SRs, which refer to matters that are essential to ship the final product.

This pressure is especially evident in project B, where the temporal occurrences of SRs of different severity are quite distinct. Also in project A such effect is remarkable, even if not intense. Project A and B are the projects with the shorter time frames (Figure 6). In projects C and D the pressure is still evident but at a much lower degree, since there is a significant overlap between the temporal occurrences of SRs of different severity. The managers at the company confirm that project A and B were the one for which the highest level of pressure was experience by the employees.

With respect to the types, it appears that in both project A and B there were initially requests related mainly to problems in understanding the requirements of the final systems –the types "Deficiency," "Software," and "Feature." In the later

stages, the requests were more related to minor issues or to understanding what the systems were supposed to do –the types "Enhancements" and "Assistance."

Also for types of SRs, the managers confirm that the results conform to the real evolution of such projects.

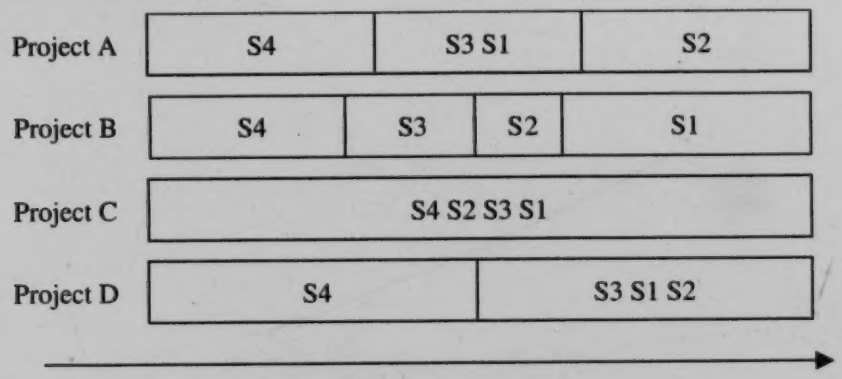


Figure 9: Gamma analysis of the severity

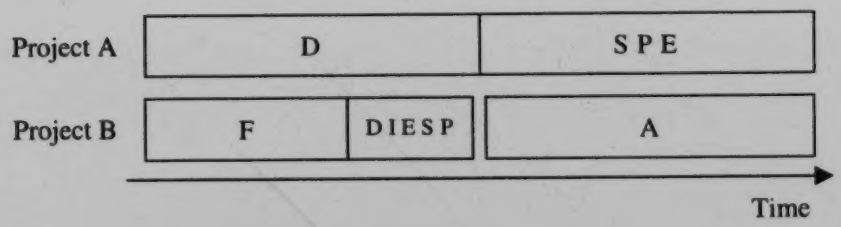


Figure 10: Gamma analysis for projects A and B

6.1.4. Sensitivity of the Models

Since the SRs are collected by developers, they are subject to human errors and imprecision. To assess sensitivity of the models to errors, a Monte Carlo simulation is run to determine what level of error affect the structure of the models and what can be tolerated without problems.

A white noise, $N(0,s)$ is added to the timing of the original data. The standard deviation, s, assumes values from half a day to an entire year. For each value of the standard deviation, the simulation is iterated 100 times. In each iteration, the

associated models are extracted and it is checked if their parameters are within the 95% confidence interval of the original models.

The graph in Figure 11 contains the results for the best performing SRGMs. The x-axis is the log of the standard deviation of the noise, in days. The y-axis represent the percentage of times the parameters of each model fall within the 95% confidence interval for the given standard deviation.

The best performing SRGMs models have different sensitivity to imprecise input. Most of the models are resistant to the noise with standard deviation below 5 days, while some models can even tolerate two weeks without significant degradation in performances. Table 12 contains the ranking of the models.

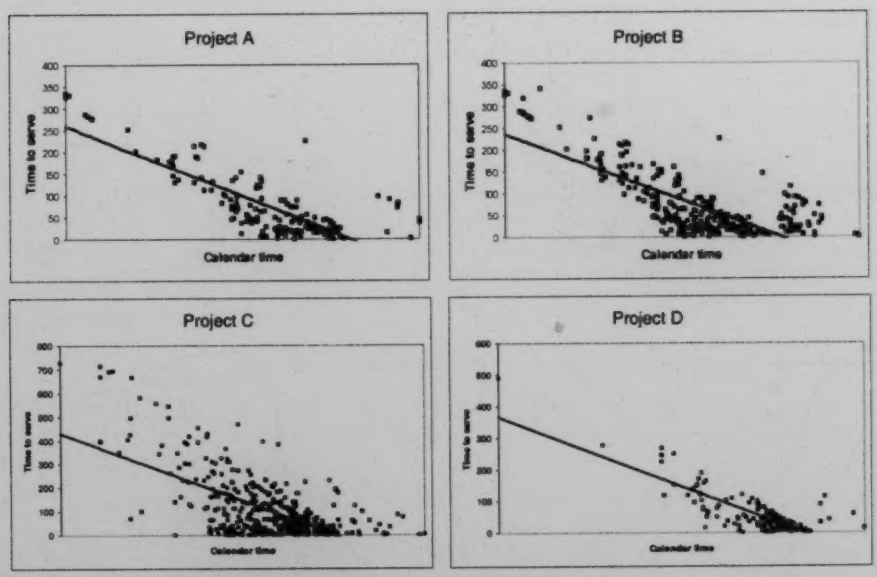


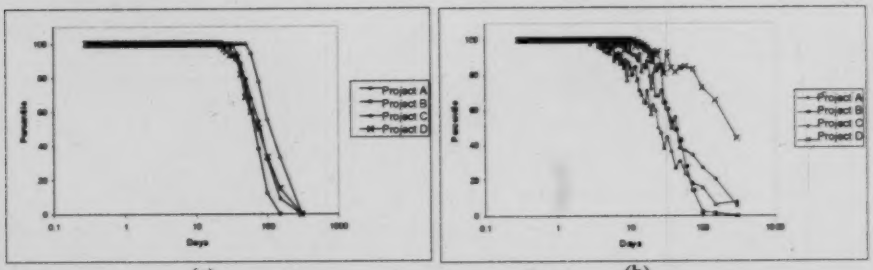
Figure 11: Performance of SRGMs in presence of error

	95% confidence intervals			
	A	*B*	*C*	*D*
1	G	HD	G	G
2	GO-S	G	HD	HD

3	L	L	L	L
4	W-S	W	N/A	N/A
5	W	GO-S	N/A	N/A
6	HD	W-S	N/A	N/A

Table 12: Ranking of the SRGMs with respect to their sensitivity to error

Linear regression showed high resistance to noisy conditions, with quite good results even for the data-entry error on the level of about 10 days, as represented in the Figure 12(a).



(a) (b)

Figure 12: Performances in presence of noise of (a) the linear regression models and (b) the gamma analysis

The gamma analysis appears more sensitive to presence of noise. The degradation starts when the error reaches about 2 days. However, the curves oscillate more: this can be a result of the non-parametric and non linear nature of this statistical analysis.

6.1.5. Summary of the Results and Discussion

In this section, the obtained results are briefly summarized and discussed.

It appears that in general SRGMs can model effectively the **timing of occurrences** of SRs provided that a multimodel approach is taken. Such multimodel approach is useful not only to achieve the different goals of the analysis but also to have more robust results for each individual goal.

The results regarding the predictive ability of the models are not completely consistent: for projects A and B there are models that can make reasonable estimations half way through the servicing process, while for projects C and D the situation is less clear and demand a closer investigation.

In this part of research, across the 4 analyzed projects, three SRGMs perform have consistently good. Such models are the Gomperz, the Hossain-Dahiya, and the Logistic.

Overall, the models do not change significantly taking into account either the information on severity of SRs or the knowledge on to which internal component the SRs refer. However, the impacts of SRs of severity S1 are unclear, probably due to the limited amount of data available.

The analysis of the time of SRs occurrences with SRGMs is quite robust to errors: in general the parameters of the best performing models do not change significantly for white, Gaussian error with standard deviation up to five days.

The **time to serve** is well correlated to the calendar time. However, the linear regressions extracted from the data satisfy only partially the criteria for a linear model. This might be due to the influence of severity of SRs, which, if taken into account, appear to alter the structure of the models. Unfortunately, also in this case, the limited amount of data does not allow to obtain more definite results.

The linear models are also quite resistant to noise; their parameters do not significantly change for levels of standard deviation up to 5 days.

The **temporal occurrences of the different kinds of SRs** can be described with gamma analysis, resulting in models that are accurate explanation of what effectively happened. For all the four projects the more severe SRs tend to occur especially at the end of development, due to the final rush to have the product shipped. This situation is especially true for project B, the one that had the most severe pressure, according to the managers of the company.

With respect to the types of SRs, SRs related to requirements occur earlier while SRs for product understanding and use later.

The models extracted with gamma analysis are more sensitive to errors. The level of tolerance is a standard deviation of about 2 days.

6.2. Discussion of the Experimental Data from Commercial Domain

This section focuses on three different software projects, referred to as Project 1 (P1), Project 2 (P2), and Project 3 (P3), coming from two companies (companies A and B). P1 is developed by company A, and the other two projects (P2 and P3) by company B. All three projects are developed for the commercial application domain.

P1 is mainly written in C++, while P2 and P3 are developed in RPG. The size of the products, expressed in the thousands of lines of code (KLOC), is 48 KLOC for P1, 99 KLOC for P2, and 100 KLOC for P3.

The developers in both companies had education level equivalent to a BSc in Electrical and Computer Engineering or Computer Science. Personnel and resources were assigned to the projects with no particular pattern, suggesting no selection bias.

Severity Level	Description	Count
Critical	A problem that makes the whole system, a subsystem, or major feature (function) non-operational.	5 (3.7%)
Serious	Partial failure of system operations with no significant effect to overall system performance.	73 (54.5%)
Non-critical	A problem that can be worked around causing no serious problems.	56 (41.8%)

Priority level	Description	Count
High	Issue is worked 24 hours a day, seven days a week until the problem is resolved.	68 (50.8%)
Medium	The objective is to resolve the reported problem within one month.	56 (41.8%)
Low	The objective is to resolve the reported problem in less than 6 months.	10 (7.4%)

Table 13: Severity and priority levels of the SRs in the P1 dataset

Type of SRs	Description	Count
Change Request (C)	Request for an additional new feature or a significant change or enhancement of the existing requirements.	30 22.4%
Documentation (Doc)	Request for a fix of incomplete or incorrect documentation.	1 0.8%
Duplicate (D)	Request that was determined to report on the same problem as an earlier SR.	7 5.2%
Mistake (M)	SR reporting on a problem that does not exist.	2 1.5%
Support (S)	Request for additional help with configuration of the system, or interpretation of the specification.	16 11.9%
S/W Bug (Sb)	System-level problem traceable to code.	78 58.2%

Table 14: Types of SRs in P1 dataset

The development of P1 took two and a half years, while development of P2 and P3 took five and five and a half years, respectively.

There were in total 134 SRs in four partially overlapped releases of P1. Project P2 had four consecutive releases with the total of 126 SRs, and P3 had five releases with the total of 76 SRs. There was practically no overlapping in the development of the successive releases of projects P2 and P3.

The information about the date the SRs occurred and the date they were fixed is recorded in both companies. Only the P1 dataset has information about severity, priority level (Table 13), and types of SRs (Table 14) associated with each SR.

The severity level of an SR is related to its importance for the overall functionality of the application from the customers' perspective. The priority of an SR depends on the organizational and technical importance of an SR. Table 7 contains description of the severity and priority levels of SRs in the P1 dataset.

Serious SRs make the major part of the dataset (54.5%), followed by non-critical SRs (41.8%), while less than 4% of all SRs are critical. Low percentage of critical SRs can be interpreted as a consequence of the application domain, but it can also be attributed to the corporate development process and subjective factors. For example, developers might be reluctant to assign critical severity to an SR due to the commitments required in fixing such an SR (see Table 7).

When the priority assigned to SRs is concerned, the majority of SRs have high priority (50.8%), followed by medium (41.8%) and low priority (7.4%).

With respect to types of SRs, those SRs referring to software "bugs" are most frequent type of SRs in the P1 dataset (58.2%). The least often type of SRs is related to documentation, making only 0.8% of the total count.

Types of SRs also report on requests that did not result in modifications to the system: duplicate and mistake SRs. However, these SRs are treated in the same way as the other types since they also require effort to be spent in identification, isolation and further investigation of the reported problem.

6.2.1. *Extraction and Analysis of the Models for timing of SRs*

In this section, the ability of models listed in Table 3 to describe the occurrence of SRs over time is analyzed.

The least square error regression is used for fitting of the parameters of the models. The statistical tool that was used employs an iterative estimation algorithm for finding the global minimum of the cost function, and the bootstrap method to determine the confidence intervals for the parameters.

The goal in modeling the occurrence of SRs is both to create an accurate description of the SRs servicing process, to serve as a base for assessment and future improvements, and to predict the occurrence of the SRs early in the software evolution.

The criteria used to evaluate SRGMs are (similar to Section 6.1): goodness of fit, accuracy of the final point, relative precision of fit, coverage of fit, and predictive ability.

The *goodness of fit* represents how well the model fits the data. This criterion is measured using the standard deviation of residuals, i.e., the root mean square error (Tryfos, 1997). The unit used for goodness of fit in Table 15 is the number of SRs.

The *accuracy of the final point* corresponds to the effectiveness of the model in determining the final number of SRs observed in the dataset. It is measured using the relative error in the estimated number of SRs, and expressed in percentiles of the final number of SRs in Table 15.

The *relative precision of fit* is the size of the 95% confidence interval for parameters of the model, normalized over the development time interval. The unit used for this measure in Table 15 corresponds to the product of the time (expressed in days) and the number of SRs. Models with low value of the relative precision of fit typically have high capability of providing useful information about the occurrence of SRs.

The *coverage of fit* represents the degree to which the bootstrap confidence interval captures the observed service requests. In Table 15, the coverage of fit is expressed in percentile of the total number of SRs within the confidence interval.

The coverage and the relative precision of fit represent complementary aspects of the model fit. To properly evaluate a model they should be considered together. A wide confidence interval captures most of the data points, but it does not provide useful information about the occurrence of SRs.

The *predictive ability* describes how early in the development process a model is able to predict the final count of SRs within the ±10% range of the final number of SRs recorded. Values in Table 15 represent the ratio of the time necessary for the model to converge and the total development time.

The three software projects come from the same application domain but are developed in different programming languages and environments, and during relatively long period of time. Consequently, it is difficult to restrict the nine SRGMs to a smaller set of models with consistently good results across the different datasets, i.e., with high applicability level. This is primarily because the occurrence of SRs over time widely departs from the clear S-shaped or concave behavior for all three software projects (Figure 13).

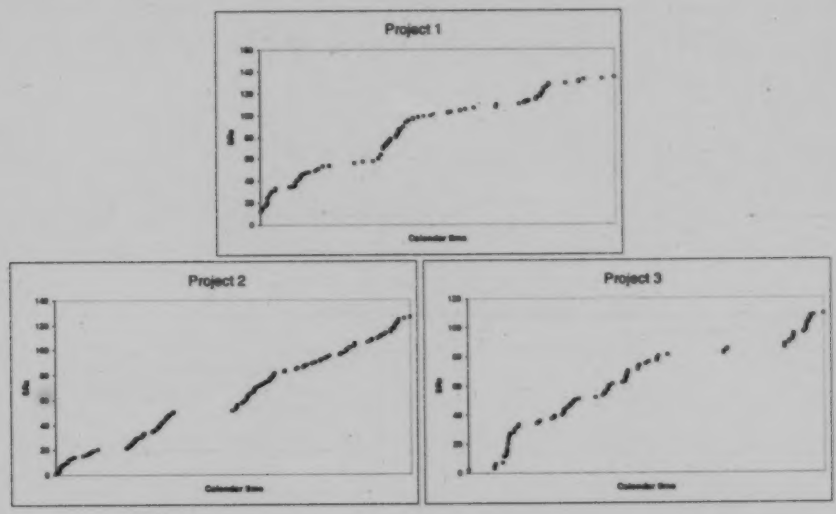


Figure 13: Occurrence of SRs over time for the three projects

From the occurrence of the SRs over time, shown in Figure 13, it is clear that P2 does not reach a stable phase with low SR rate. Reasons for releasing the product before the stable stage in servicing SRs is reached depend on other aspects of the software development process, corporate release methodology, and various business pressures. This kind of trend cannot be successfully explained by any of the available models, since all of them assume a finite number of errors present in the software.

Goodness of fit, accuracy of the final point, relative precision of fit, and coverage of fit are all used to quantify various aspects of the models' applicability, capability, and effectiveness. Predictive ability is used to validate predictive characteristics of the model.

The accuracy of the final point and the predictive ability are both associated with the precision in estimation of the final count of SRs in the dataset. Consequently, the rankings of the models with respect to these two criteria are interrelated. When occurrence of SRs does not reach the stable phase, e.g., in case of P2, the results of both criteria do not make sense. Therefore the respective columns are grayed for P2.

Results of the analysis and model assessment are presented in Table 15, with the best results shown in bold. In Table 16, the models are ranked based on their performances.

	Goodness of Fit			**Accuracy of Final Point**			**Relative Precision of Fit**		
	P1	*P2*	*P3*	*P1*	*P2*	*P3*	*P1*	*P2*	*P3*
G	5.44	4.00	4.99	5.22	47.23	5.30	144	237	158
GO	8.82	4.14	4.80	5.51	566.24	16.02	260	2518	331
GO-S	13.50	6.88	5.87	8.74	17.57	8.34	172	282	205
HD	8.82	4.21	4.80	5.52	447.46	16.01	258	811	387
L	5.74	4.56	5.48	2.55	15.19	9.39	488	142	252
W	5.83	4.35	4.65	4645.03	209.28	3.45	1625	3610	1874
W-S	6.07	4.23	5.11	513.92	923.64	3.02	1664	2604	2435

YE	8.79	4.34	4.85	1480.28	711.33	129.69	579	4709	1927
YR	15.32	8.94	7.48	0.01	$1.36\ 10^{-12}$	$1.55\ 10^{-12}$	300	5343	8181

	Coverage of Fit			Predictive Ability		
G	*P1*	*P2*	*P3*	*P1*	*P2*	*P3*
GO	59.30	90.32	57.29	0.67	N/A	0.56
GO-S	66.28	95.97	82.29	0.76	N/A	N/A
HD	15.12	63.71	11.46	0.91	N/A	0.97
L	66.28	83.87	80.21	0.76	N/A	N/A
W	33.72	58.87	14.58	0.76	N/A	0.97
W-S	97.67	99.19	96.88	N/A	N/A	0.95
YE	96.51	94.35	98.96	N/A	N/A	0.45
YR	52.33	99.19	96.88	N/A	N/A	N/A
	61.63	99.19	98.96	0.67	0.25	0.08

Table 15: Result of the analysis of the SRGMs for the three projects

	Goodness of Fit			Accuracy of Final Point			Relative Precision of Fit		
	P1	*P2*	*P3*	*P1*	*P2*	*P3*	*P1*	*P2*	*P3*
1	G	G	W	YR	YR	YR	G	L	G
2	L	GO	HD	L	L	W-S	GO-S	G	GO-S
3	W	HD	GO	G	GO-S	W	HD	GO-S	L
4	W-S	W-S	YE	GO	G	G	GO	HD	GO
5	YE	YE	G	HD	W	GO-S	YR	GO	HD
6	GO	W	W-S	GO-S	HD	L	L	W-S	W
7	HD	L	L	W-S	GO	HD	YE	W	YE
8	GO-S	GO-S	GO-S	YE	YE	GO	W	YE	W-S
9	YR	YR	YR	W	W-S	YE	W-S	YR	YR

	Coverage of Fit			Predictive Ability		
	P1	*P2*	*P3*	*P1*	*P2*	*P3*
1	W	W	W-S	G	YR	YR
2	W-S	YE	YR	YR	G	W-S
3	HD	YR	W	GO	GO	W
4	GO	GO	YE	HD	GO-S	GO-S

5	YR	W-S	GO	L	HD	L
6	G	G	HD	GO-S	L	G
7	YE	HD	G	W	W	GO
8	L	GO-S	L	W-S	W-S	HD
9	GO-S	L	GO-S	YE	YE	YE

Table 16: Ranking of the SRGMs for the three projects

In general, the G and the L models show relatively high level of applicability by consistently ranking high on the list of models' performance with respect to the set of criteria used in this study. The G model is especially good in terms of relative precision and goodness of fit. On the other hand, the YR model demonstrates a fair accuracy in estimation and early prediction of the final point.

For all the three projects the YE model performs poorly in most of the criteria: accuracy of the final point, relative precision of fit, and predictive ability. This ranks the YE model relatively low on the overall list of the applicable models together with the GO-S and the YR models, both of which have consistently poor goodness of fit, while the GO-S model also has low coverage of fit.

Although the G and the GO-S models have good precision of fit for all the projects, they also show relatively poor results with respect to the coverage of fit. On the other hand, the W model is highly ranked for all three datasets with respect to the coverage of fit, having much lower ranking when the relative precision of fit is concerned.

As mentioned, relative precision and coverage of fit correspond to complementary aspects of the model's performance. This study is interested in both capturing the shape of the occurrence of SRs over time, and covering most of the observed data points within the confidence interval. Although a very wide confidence interval captures most of the data points, it does not provide useful information about the occurrence of SRs. A desirable descriptive model should have a low value for the relative precision of fit and a high value for the coverage of fit.

Figure 14 presents scatterplot of the coverage of fit and the relative precision of fit (on the logarithmic scale) for the nine models and the three datasets. The high

value of Spearman's non-parametric correlation between the two measures ($r = 0.83$, $p < 10^{-3}$) indicates that the models with a high relative precision typically have high coverage of fit.

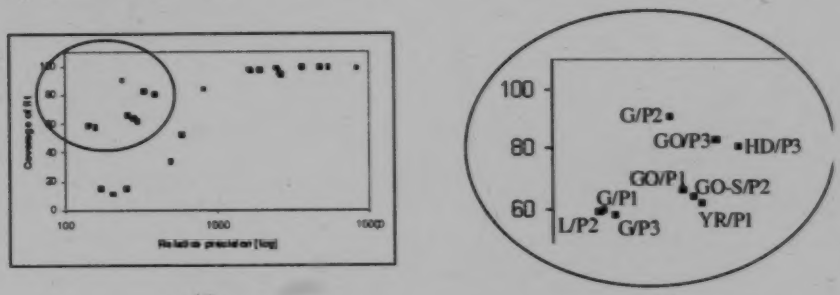


Figure 14: Scatterplot of the coverage of fit vs. relative precision of fit (a), and the zoomed portion of this graph (b)

The models with low values of the relative precision and high values of the coverage of fit are located in the upper left-hand side of the graph (a) in Figure 14. This portion of the graph is zoomed in Figure 14 (b), showing the best models and the corresponding datasets with respect to both precision and coverage fit.

6.2.2. Time to Resolve the SRs

Another important aspect in the description of the process of serving SRs is the productivity of this process. Productivity can be expressed in terms of the time required to service an SR after the request was submitted. It is intuitive to expect that the learning process, i.e., increasing understanding of the system as time progresses, has a strong impact to the productivity in servicing SRs. However, it is interesting to check if there are other aspects with high influence to the time required for servicing SRs.

This analysis is performed only on the P1 dataset since the time to service SRs is not recorded in the other two datasets.

For modeling the effect of other categorical SR attributes, such as severity, priority level, and software release, the stepwise linear regression is used with dummy variables (Tryfos, 1997).

The resulting model (Table 11) identifies the calendar time as the only statistically relevant variable with influence on variation in the servicing time. Quadratic and cubic models have also been tried but without any improvement in the performance of the final model.

Slope	Slope Signif.	Intercept	Interc. Signif.	*r*
-0.12	0.01	121.14	$p < 10^{-03}$	0.22

Table 17: Coefficients of linear regression

The resulting linear model has a low correlation coefficient and a moderate value of the slope parameter. In addition, there is a clear pattern in the distribution of residuals for this regression (Figure 15).

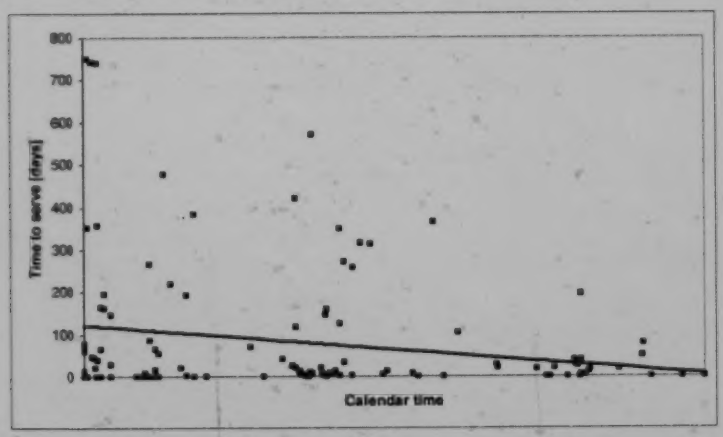


Figure 15: Variation of time to service SRs with calendar time

6.2.3. Kinds of SRs

Understanding the distribution and occurrence of the different types of SRs over time allows a proper allocation of the right kind of resources and better insight in the evolution of the servicing process.

As mentioned, severity level of an SR is related with the customers' perspective of system's functionality, while the priority is assigned according to SR's technical and organizational importance in the development process. While

intuitively these two aspects of SRs are interrelated, it is interesting to analyze this relation in more details.

Figure 16 graphically presents the percentage in which each priority level contributes to a total count of SRs, across the different severity levels. From this graph, it can be seen that all critically severe SRs were assigned high priority level. The percentage of SRs assigned both low and medium priority levels decreases with increased severity levels, while the percentage of high-priority SRs increases.

To some extent, SRs with higher severity correspond to SRs with higher priority level. However, the correlation coefficient for categorical data (Freund and Simon, 1996), calculated with respect to the severity and priority levels of SRs, has a moderate positive value r = 0.33.

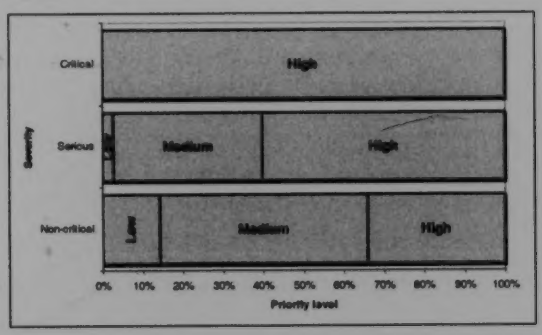


Figure 16: Severity and priority levels of SRs for the P1 dataset

To describe occurrence of different types of SRs over time, gamma analysis is performed with respect to severity, priority level, and type of SRs from P1. Phases with less than 4 data points in the dataset are not considered for this analysis (Pelz, 1985).

With respect to severity of SRs, Figure 10 shows that non-critical SRs tend to appear early in the process, with a low level of overlapping with the following phase in which critical and severe SRs appear.

While there are two distinctive phases in occurrence of SRs with different severities, the SRs with different priority levels tend to be more overlapped

(Figure 10). Generally, high priority SRs tend to appear earlier in the process, followed by SRs with low and medium priority.

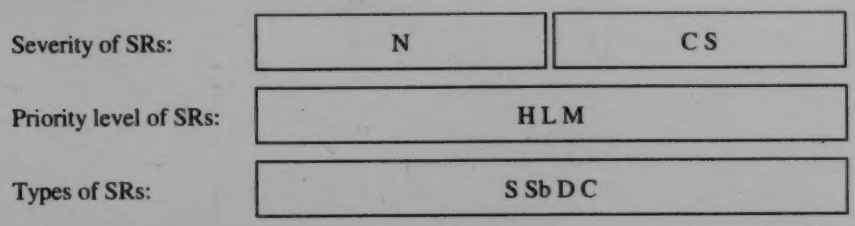


Figure 17: Gamma analysis for P1 with respect to the severity, priority level, and types of SRs

Occurrence of different types of SRs also shows high level of overlapping. Requests for support (S) tend to appear first, followed by software bugs (Sb) and duplicate SRs (D), while requests for change (C) occur later in the process.

6.2.4. Sensitivity of the Models

The data about SRs are subject to human errors and imprecision in the collection process. A Monte Carlo simulation with additive white noise is performed to determine the level of error that can be tolerated without significantly affecting models' structure.

The white noise is added to the timing of the original data, with standard deviation ranging from half a day up to an entire year. The simulation is repeated 100 times for each value of the standard deviation. Every time the simulation is run, the parameters of the models are calculated and the check is performed whether these parameters are within the 95% confidence interval of the original models.

The sensitivity to imprecision in the input data depends both on the underlying dataset and on the corresponding model. Models applied to all of the available datasets show relatively good resistance to errors in the input data ranging from around 9 days for P2 (the G model), up to 19 days for P3 (the W model). Figure 18 presents the results of Monte Carlo analysis and Table 12 contains the ranking of the models with respect to their robustness. In Figure 18 the x-axis is the log of the standard deviation of the noise, expressed in days. The y-axis represents the

percentage of the total number of simulation runs, for the given standard deviation, with the parameters of the resulting model within the confidence interval.

The GO-S model consistently shows high robustness to the introduced noise for all the projects. The other six models follow this pattern, while the YE and the YR models show poor resistance to noise.

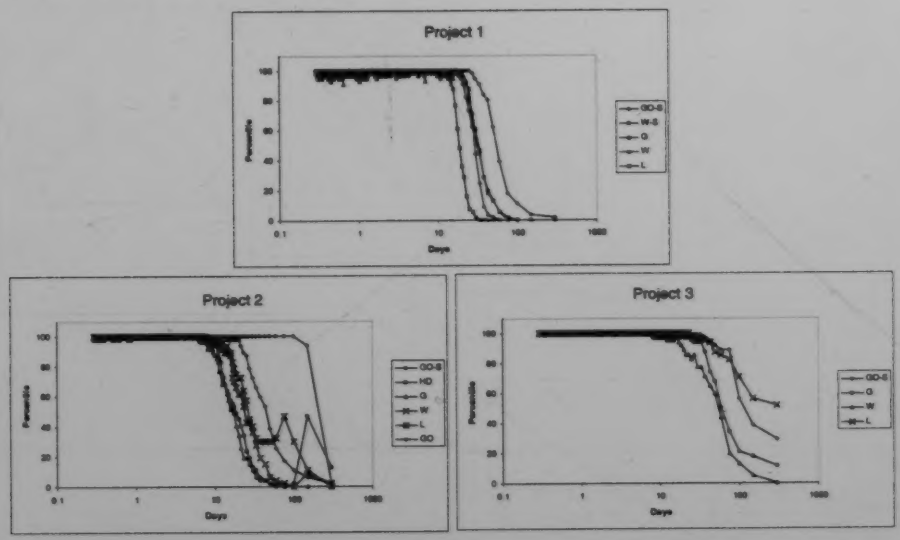


Figure 18: Monte Carlo analysis of the robustness of SRGMs to the noise in the input data

A model with wide confidence interval for its parameters, i.e., high value of the precision of fit, typically has relatively high resistance to noise. Such a model, however, does not provide useful information about the occurrence and the process of servicing SRs.

	P1	P2	P3
1	GO-S	GO	GO-S
2	W	GO-S	L
3	G	W	G

	P1	P2	P3
4	W-S	L	W
5	L	HD	
6		G	

Table 18: Ranking of the SRGMs with respect to the robustness to noise in the input data

Monte Carlo analysis is also used to assess the robustness of the linear model applied to time to serve SRs. This model also shows high robustness to imprecision in the input data, being able to tolerate noise corresponding to approximately two weeks.

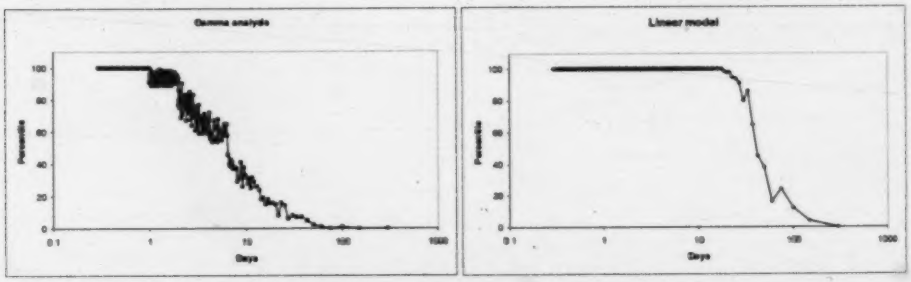


Figure 19: Robustness of the gamma analysis linear regression results in presence of the noise in the input data

Results of the gamma analysis appear to be more sensitive to the presence of imprecision in the input data. The degradation starts with the errors in the range of one day. The graphs in Figure 19 present the results of the Monte Carlo simulations for gamma analysis and for the linear regression model.

6.2.5. Summary of the Results and Discussion

This study is focused on analysis of the data from the commercial application domain. It is interesting to compare results of this analysis with the findings in other domains. A similar analysis of service requests in the real-time applications

is presented in Section 6.1. Overall, this study shows more heterogeneous results caused by diverse profiles in occurrence of SRs in the three analyzed projects.

As already mentioned, the process of servicing SRs is strongly related to the reliability issues since SRs often represent reports of software failures. Clearly, reliability requirements are considerably different for the two application domains. While reliability is an important issue in delivering a commercial application, it is *the essential factor* in real-time applications.

Unlike the analysis of the real-time projects, where it was possible to distinguish three models performing reasonably well across the analyzed projects and with respect to the set of criteria employed, in case of the commercial projects analyzed in this section, it is much more difficult to identify the most applicable models. To some extent the G and the L models perform better than other models since they show high flexibility and effectiveness across the analyzed projects. On the other hand the YE model performs relatively poorly for all the datasets used in this study.

The results evidence much lower impact of the learning process on the occurrence of SRs over time in the commercial domain compared to real-time applications. Cumulative occurrence of SRs over time tends to be more concave in case of commercial applications, indicating less critical initial learning process. Resulting linear regression model also supports shorter initial time required for serving SRs in commercial applications and much lower correlation with the calendar time.

This analysis has access to information about severity and priority level assigned to SRs. Due to the characteristics of the application domain, the percentage of critically severe SRs is lower compared to the real-time applications analyzed in Section 6.1. Results of gamma analysis showed two distinctive phases in the occurrence of SRs with different severity, while the SRs with different priority levels tend to be more overlapped. Distribution of SRs over time with respect to severity indicates similarity with the one in the real-time projects. In both cases, SRs with lower severity tend to appear earlier in the process. Possible reason for this is that more focus is put on critical SRs towards the release deadline due to

the business pressures. Although higher priority SRs tend to appear earlier in the process, the level of overlapping is high.

While different classifications of SR types make it impossible to make direct comparison with respect to the occurrence of SRs in the two domains, a higher level of overlapping of the different phases can be identified in the commercial applications compared to the real-time domain projects discussed in Section 6.1.

The results of the Monte Carlo simulation facilitate assessment of the level of human error present in the collected data that can be tolerated by the models. With respect to the robustness of the SRGMs to the imprecision in the input data, results are rather consistent with the previous work. Most of the models (excluding the YE and the YR models) show relatively low sensitivity to noise in the range of approximately one week or more. The G model is the best choice in terms of robustness in the real-time domain. The GO-S is consistently performing well concerning this aspect in commercial applications. Sensitivity to noise of both gamma analysis and linear regression model is also similar in both domains. Linear model applied to time to service SRs is typically resistant to the errors in the input data of the order of approximately 10 days. Gamma analysis appears to be more sensitive method for which degradation starts with the errors in the range of one day.

Summary of the findings in this study and comparison with the analysis performed in the real-time telecommunication domain is provided in Table 19.

	Real-time	**Commercial**
Occurrence of SRs over time	Clearly S-shaped	Less clear, concave
Highly applicable SRGMs	G, HD, L	G, L
Less applicable SRGMs	GO, YE, YR	YE
Robustness to	Approx. 5 days	Approx. 7 days

	Real-time	Commercial
imprecision		
Impact of learning to time to solve SRs	High	Low
Occurrence of SRs by severity	Low severity SRs occur early	Low severity SRs occur early
Occurrence of SRs by types	Relatively distinct phases	Highly overlapped phases

Table 19: Comparison of the results in real-time and commercial domain

7. Impact of Object-Oriented Design on Class Defect Behavior

7.1. Discussion of the Experimental Data from Real-Time Domain

This section focuses on five projects developed by a North-American company that prefers to remain anonymous. The projects were developed and tested over approximately five years. The projects are developed for embedded system in a real-time telecommunication domain, mainly using the C++ programming language.

Developers assigned to the projects had similar experience and education levels equivalent to BSc in Electrical and Computer Engineering or in Computer Science. No particular pattern was observed in the assignment of developers to projects, suggesting no selection bias.

Total size of the five projects is 63394 lines of code, with 395 classes for which set of object-oriented design metrics was extracted and the number of revisions was recorded (Table 20).

The CK suite of object-oriented design metrics and the source lines of code counts are collected from the source code using WebMetrics, a software metrics collection system (Succi *et al.*, 1998).

Project	Number of classes	LOC
A	93	8802
B	119	6496
C	101	24989
D	44	5316
E	38	17791

Table 20: Number of available data points for projects

A summary of the descriptive statistics for the extracted metrics is provided in Table 21. It is found that both metrics dealing with inheritance, the depth of inheritance tree and the number of children assume low values. Similar behavior was also observed in work by Chidamber *et al.* (1998) and by Ronchetti and Succi (2000).

	Project A				**Project B**			
	Min	*Max*	*Mean*	*Std. Dev.*	*Min*	*Max*	*Mean*	*Std. Dev.*
LOC	2	754	94.65	129.11	1	1128	57.24	156.86
NOM	0	54	9.78	9.29	0	74	9.24	13.79
DIT	0	4	0.90	1.27	0	3	0.97	1.12
NOC	0	11	0.27	1.39	0	5	0.16	0.62
CBO	0	68	11.68	12.17	0	63	4.17	8.02
RFC	0	122	24.59	25.93	0	246	17.59	35.36
LCOM	0	821	59.44	115.94	0	1540	87.90	234.98
Defects	1	28	5.28	4.71	0	22	2.35	5.28
	Project C				**Project D**			
	Min	*Max*	*Mean*	*Std. Dev.*	*Min*	*Max*	*Mean*	*Std. Dev.*

LOC	1	1377	247.42	302.98	2	1674	120.82	258.24
NOM	1	219	32.60	41.13	0	94	16.09	19.26
DIT	0	2	0.97	0.96	0	1	0.25	0.44
NOC	0	11	0.16	1.20	0	4	0.14	0.63
CBO	0	121	23.17	24.09	0	111	11.05	18.24
RFC	2	283	67.46	71.90	0	336	33.23	52.38
LCOM	0	23247	1041.30	3197.59	0	3897	256.57	701.88
Defects	0	24	1.38	3.12	0	16	1.30	2.92
	Project E							
	Min	*Max*	*Mean*	*Std. Dev.*				
LOC	2	3181	468.18	611.86				
NOM	0	188	41.61	43.41				
DIT	0	2	0.26	0.55				
NOC	0	2	0.05	0.32				
CBO	0	124	17.82	22.67				
RFC	0	418	69.05	75.19				
LCOM	0	17048	1606.03	3585.45				
Defects	0	32	5.55	7.03				

Table 21: Descriptive Statistics of the extracted metrics and defects for the projects

External measure used as the dependent variable in this analysis is the number of revisions for a class. As commonly done in software engineering, it is assumed here that number of revisions closely corresponds to the number of defects found in a class. In this study, these two terms will be used interchangeably.

The number of defects found in a class ranges from 0 to 32, with most of the classes having very low number of defects. Boxplots of the observed data (Figure 20) evidence the non-normal, Poisson-like, nature of the distributions. The number of defect counts for project D is low resulting in the boxplot for this project to be concentrated around zero.

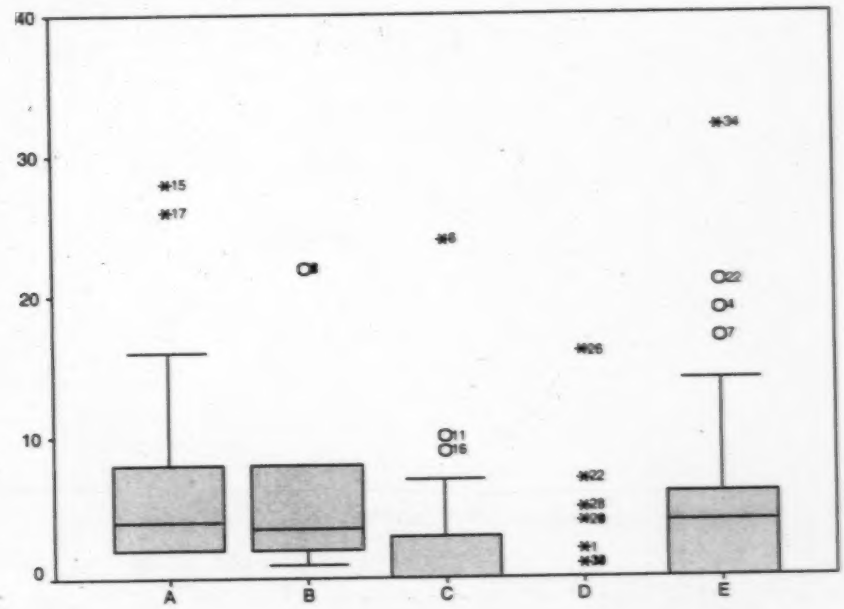


Figure 20: Boxplots for distribution of the number of defects

As observed by Ebert and Baisch (1998), the distribution of defects often follows the Pareto rule. Alberg diagrams (Ohlsson and Alberg, 1996) evidence presence of this principle, that is relatively small portion of the system classes is causing most of the defects. Figure 21 shows the observed number of defects for the five projects in the form of the Alberg diagram. From the figure, it can be seen that 80% of the defects are caused by only 3% of the classes in projects B, C, and E, 4% in project E, and 28% in project A.

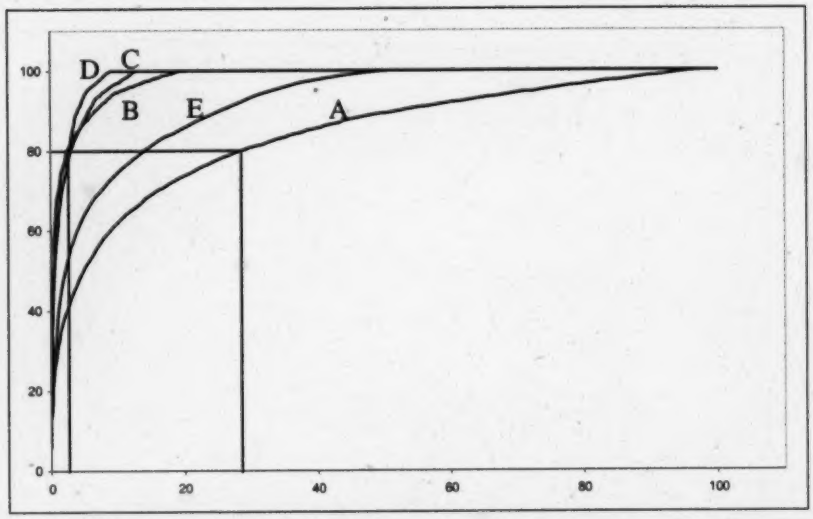


Figure 21: Cumulative number of defects for the five projects

7.1.1. Extraction of the Models

To determine models capturing the number of defects on the basis of object-oriented metrics, the analysis proceeds as follows. The Spearman rank correlation is calculated between the number of defects and the internal metrics. Table 22 presents the correlation coefficients with those statistically significant at the 10^{-3} level marked with the star. Then, more advanced statistical models are applied to deal with non-normal distribution of the dependent variable and other specifics of the empirical data. For this purpose, PRM, NBRM, and ZINBRM based on the extracted set of design metrics for modeling the number of defects are used. In these models, the exponent of the regression function is a linear combination of the metrics used as predictors.

	CBO	DIT	LCOM	NOC	NOM	RFC	LOC
A	0.00	-0.07	0.11	-0.20	0.14	0.11	0.08
B	0.36*	0.59*	0.24	-0.13	0.22	0.31	0.24
C	0.43*	0.24	0.46	0.031	0.41*	0.43*	0.46*

	CBO	DIT	LCOM	NOC	NOM	RFC	LOC
D	0.19	0.05	0.16	-0.17	0.18	0.25	0.24
E	0.45	0.06	0.32	-0.21	0.31	0.45	0.52

Table 22: Correlations between number of defects and internal metrics for the projects

The regression function in the PRM can be written in the log-linear form:

$$ln(\mu) = x\beta$$

This suggests that the PRM can be approximated by the linear regression model:

$$ln(y) = x\beta+\varepsilon$$

Since y_i can also assume zero value, it is necessary to add a positive constant c to the dependent variable before taking the log (Long, 1997). Values of c equal to 0.1 and 0.5 are typically used. In this analysis, $c = 0.5$ is used. The resulting regression model is given with:

$$ln(y+c) = x\beta+\varepsilon$$

The parameters of this generalized linear model are estimated using the method of ordinal least squares (OLS). Although estimations of this model can be biased, they can be used for approximation of the statistical significance of parameters in the corresponding PRM (King, 1988).

The application of the ordinal least squares (OLS) method for fitting the parameters of the generalized linear regression model is justified if error distribution is assumed to be normal. In this case the OLS and the maximum likelihood (ML) estimates of β are approximately the same for the linear model. On the other hand, the ML provides a very general solution for fitting of the model parameters. The ML estimator is consistent, *i.e.*, the probability that the ML estimator differs from the true parameter by an arbitrary small amount tends toward zero as the sample size grows. The variance of the ML estimator is the smallest possible among consistent estimators. This feature is usually referred to

as asymptotic efficiency of the ML estimator. Thus, the ML estimators are used for the models in this analysis.

Log-likelihood function for the ML estimation in case of the Poisson distribution is given by:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{k} (y_i \cdot ln(\mu_i(\boldsymbol{\beta})) - \mu_i(\boldsymbol{\beta}))$$

By maximizing this sum over the available data points the vector of model parameters $\boldsymbol{\beta}=(\beta_1, ..., \beta_n)$ is determined.

Instead of the OLS, the more general ML method for estimation of the PRM, NBRM, and ZINBRM using the five predictors with highest correlation coefficients in from Table 22. In the ZINBRM, the logit model is used for prediction of zeroes in combination with the NBRM. Due to the high cross correlation among the predictors and relatively small datasets, only univariate models are used.

The NBRM model can be estimated by the maximum likelihood method, maximizing the likelihood equation:

$$L(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{x}) = \prod_{i=1}^{N} Pr(y_i \mid \boldsymbol{x}_i)$$

The corresponding maximum likelihood method is also available for zero-inflated models (Long, 1997).

Table 23 presents a summary of the correlation coefficient r and dispersion parameter α for the resulting models. For comparison purposes the models are built based on each of the identified predictors. In order to compare the ability of the design metrics, models using LOC as a predictor are also included.

		RFC		NOM		LCOM		CBO		LOC	
		r	α	r	α	r	α	r	α	r	α
A	PRM OLS	0.512	17.31	0.433	0.33	0.505	2.75	0.099	8.29	0.085	11.39
	PRM ML	0.349	0.32	0.398	0.33	0.495	0.30	0.182	0.36	0.171	0.35

		RFC		NOM		LCOM		CBO		LOC	
		r	*α*	*r*	*α*	*r*	*α*	*r*	*α*	*r*	*α*
	NBRM	0.342	0.32	0.278	0.11	0.475	0.30	0.182	0.34	0.171	0.35
	ZINBRM	0.342	0.32	0.276	0.02	0.476	0.23	0.182	0.34	0.171	0.17
B	PRM OLS	0.392	0.96	0.177	2.11	0.012	9.79	0.208	0.27	0.023	33.30
	PRM ML	0.391	0.45	0.154	0.26	0.070	0.25	0.194	0.17	0.112	0.83
	NBRM	0.392	0.32	0.110	0.07	0.078	0.84	0.108	0.83	0.127	0.05
	ZINBRM	0.297	0.29	0.122	0.05	0.072	4.24	0.066	4.01	0.121	0.05
C	PRM OLS	0.548	4.87	0.213	3.45	0.292	2.86	0.733	2.98	0.799	27.84
	PRM ML	0.548	4.18	0.193	2.96	0.267	2.49	0.655	0.46	0.831	2.96
	NBRM	0.550	3.33	0.264	0.31	0.817	0.60	0.677	0.39	0.831	2.52
	ZINBRM	0.551	2.91	0.154	0.16	0.823	0.46	0.670	0.31	0.832	1.34
D	PRM OLS	0.156	11.02	0.222	5.55	0.173	7.22	0.168	4.73	0.151	5.82
	PRM ML	0.190	7.50	0.221	5.55	0.204	0.69	0.189	0.57	0.162	5.64
	NBRM	0.175	5.62	0.219	0.54	0.195	0.60	0.189	0.76	0.161	3.93
	ZINBRM	0.175	5.44	0.231	0.48	0.196	0.43	0.189	0.76	0.161	0.62
E	PRM OLS	0.648	1.28	0.262	1.46	0.398	1.37	0.727	1.13	0.719	1.10
	PRM ML	0.648	1.19	0.246	1.36	0.394	1.28	0.725	1.05	0.716	1.03
	NBRM	0.648	0.50	0.226	0.10	0.389	0.12	0.717	0.79	0.708	0.77
	ZINBRM	0.648	0.50	0.230	0.03	0.391	0.04	0.719	0.59	0.711	0.57

Table 23: Performance of the models with respect to correlation and dispersion

7.1.2. *Analysis of the Results*

As explained in the previous section, the log-linear PRM is given with:

$$NumberOfDefects = e^{\beta_0 + \beta_1 x}$$

Where x is the corresponding predictor from Table 22, and β_0 and β_1 are model parameters.

PRM model is estimated using both OLS and ML methods, and NBRM and ZINBRM are built using ML. Results in Table 23 suggest significant differences compared to the baseline PRM OLS models. First, a high level of overdispersion is present in the PRM OLS models. This result is not surprising for the software metrics data. Overdispersion is significantly reduced with ML estimators. However, the other typical result of the ML method is slightly lower correlation between the observed data and the models' estimations.

Models based on the negative binomial distribution show even higher capability of dealing with overdispersion. This improvement is partially the result of increased probability of low and high values of the dependent variable with the NB distribution (Figure 2).

Although the NBRM deals with overdispersion more successfully than PRM, the disadvantage is slightly lower resulting correlation with the observed data in most of the cases.

Overall, ZINBRM results in the lowest dispersion parameters with lower correlation coefficients, similar to the NBRM. This model incorporates the capability to successfully predict zero values with the ability of the NB distribution to account for overdispersion.

Since the metrics are non-negative, positive values of model parameters suggest positive influence of the predictors to the dependent variable. The link between the dependent variable and predictors is exponential in the analyzed regression models.

Except for the project D, all resulting models suggest high influence of communication between classes, measured by RFC, to the dependent variable in

the analysis. In related studies (Table 31), only Basili *et al.* (1996) also report significant influence of RFC on the dependent variable in their study – fault probability.

RFC includes methods called from outside the class representing a measure of communication between classes. The complexity of a class also increases with the increased number of methods invoked from a class. Testing and fixing of such classes requires more effort and understanding from both testers and developers.

RFC is a measure with a relatively wide range of values, opposed to DIT or NOC for example. In this study, RFC ranges from 0 up to 418. Clearly, the higher the communication of a class with other classes, the higher will be the probability of introducing defects and, consequently, higher the need for modifications in that class.

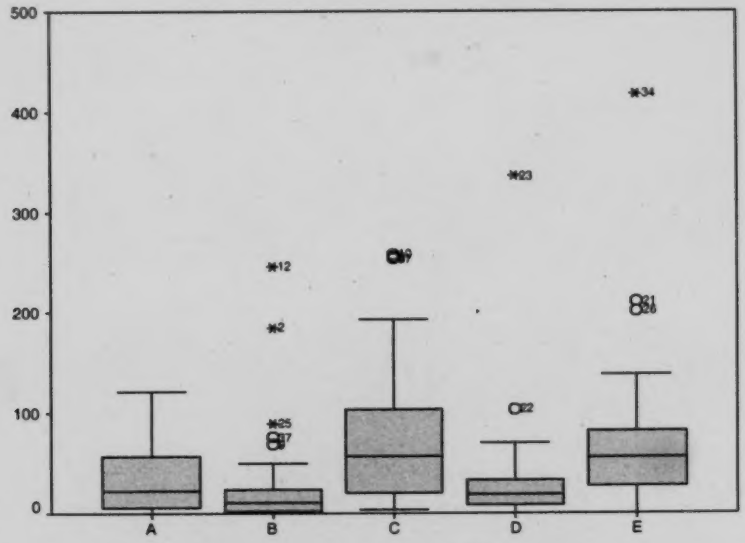


Figure 22: RFC boxplots for the five projects

The data from all five projects show that most of classes call relatively small number of methods (Figure 22). Median values of RFC are shown in Table 24.

Project A	Project B	Project C	Project D	Project D
15	6	37	20	56

Table 24: Median values of RFC

Low number of classes with high RFC values also supports the fact that small number of classes accounts for most of the defects in the system. Managing this aspect of object-oriented design is important for successful allocation of testing and fixing effort. To further investigate this issue, Alberg diagrams are used.

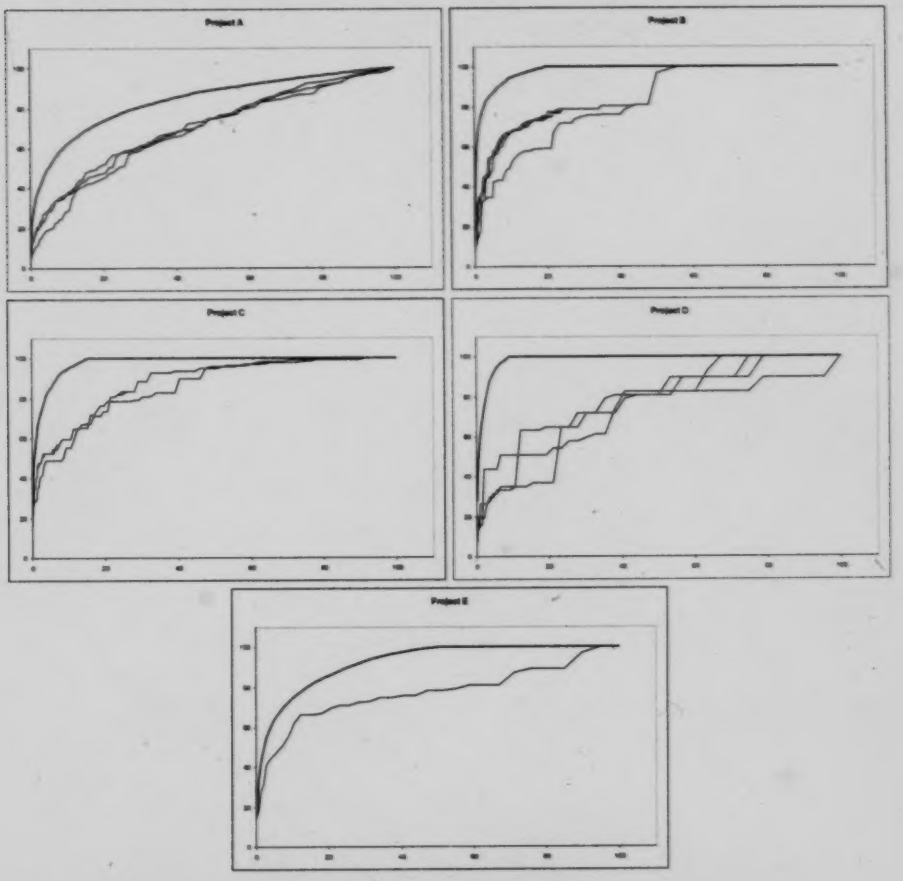


Figure 23: Alberg diagrams for the RFC-based models

As already mentioned in Section 4.6.2, Alberg diagram can be used to graphically compare performance of the different models with respect to criticality prediction (Ohlsson and Alberg, 1996). For illustration, Figure 23 shows Alberg diagrams

for the five projects using models based on RFC. Performance of the models applied to the data from project E is practically overlapped on the corresponding Alberg diagram.

Table 25 compares the percentage of the classes associated to 80% of the defects in the system with estimation from each of the models applied to the project data. The ideal number of defects for classes represents the observed best possible prediction that any model can achieve. From this value, it can be seen that very low percent of classes cause 80% of the total number of defects in the system. All the developed models model this aspect in the class criticality with limited success. However, most of the models can be used to select 50% or less of all classes for more thorough investigation, providing useful guidelines in resource allocation.

Table 25 indicates only minor difference in performance of the different models based on the same predictor and applied to the same projects. In fact NB and ZINB slightly outperform PRM model in some cases. This is an evidence that these models provide at least the same performance in indication of the defect prone classes as the other models, while providing much better description of overdispersed data with excess of zeroes.

<table>
<tr><th></th><th></th><th>Proj. A
%</th><th>Proj. B
%</th><th>Proj. C
%</th><th>Proj. D
%</th><th>Proj. E
%</th></tr>
<tr><td></td><td>Ideal</td><td>28</td><td>3</td><td>4</td><td>3</td><td>14</td></tr>
<tr><td rowspan="4">RFC</td><td>PRM OLS</td><td rowspan="4">58</td><td rowspan="4">44</td><td rowspan="2">30</td><td>38</td><td rowspan="4">59</td></tr>
<tr><td>PRM ML</td><td>44</td></tr>
<tr><td>NBRM</td><td rowspan="2">22</td><td rowspan="2">41</td></tr>
<tr><td>ZINBRM</td></tr>
<tr><td rowspan="2">CBO</td><td>PRM OLS</td><td rowspan="2">69</td><td rowspan="2">49</td><td rowspan="2">26</td><td rowspan="2">53</td><td rowspan="2">34</td></tr>
<tr><td>PRM ML</td></tr>
</table>

		Proj. A %	Proj. B %	Proj. C %	Proj. D %	Proj. E %
	NBRM					
	ZINBRM					
LCOM	PRM OLS	58	38	24	41	55
	PRM ML					76
	NBRM	57				55
	ZINBRM					
NOM	PRM OLS	62	42	25	60	59
	PRM ML	60				
	NBRM					
	ZINBRM		41			76
LOC	PRM OLS	58	39	18	35	40
	PRM ML					
	NBRM					
	ZINBRM					

Table 25: Percentile of classes identified for inspection to detect 80% of the defects

7.2. Discussion of the Experimental data from Commercial Domain

This study is focused on two software projects from the commercial application domain. Both projects are developed in C++ programming language; project A over two and a half years, and project B over 6 years. All the developers, approximately 50 developers for project A and 11 developers for project B. All developers had similar experience and education levels (equivalent to BSc in Electrical and Computer Engineering or Computer Science) were involved in the

development process. No particular pattern was observed in the assignment of developers to different problems or parts of the system, suggesting no selection bias.

Project A consists of 150 classes and project B of 144 classes for which the set of object-oriented design metrics was extracted and the number of defects and changed LOC was recorded. Total size of the projects A and B is 23 and 25 KLOC (thousands of lines of code), respectively.

Summary of the descriptive statistics for the extracted metrics is provided in Table 26.

	Project A				**Project B**			
	Min	*Max*	*Mean*	*Std. Dev.*	*Min*	*Max*	*Mean*	*Std. Dev.*
CBO	0	111	9.95	14.43	0	23	4.69	4.70
DIT	0	2	0.35	0.56	0	11	2.39	2.47
LCOM	0	20710	428.91	2102.46	0	22138	170.94	1124.19
NOC	0	5	0.25	0.84	0	20	0.52	1.55
NOM	0	205	16.38	26.07	0	212	13.47	16.45
RFC	0	336	32.08	50.53	0	218	23.42	25.96
LOC	1	1674	105.7	227.28	1	710	55.79	86.23
Defects	0	7	0.41	0.91	0	41	1.17	3.14

Table 26: Descriptive Statistics of the extracted CK metrics and LOC for project A

Number of modifications directly related to software defects for a class is used as a proxy for the defect-proneness of the class. The models are built for this external measure as the dependent variable and metrics from the CK set as predictors.

The correlation between the collected design metrics and the number of defects is calculated and summarized in Table 27.

	CBO	DIT	LCOM	LOC	NOC	NOM	RFC
Project A	0.18	0.12	0.16	0.12	-0.12	0.14	0.17
Project B	0.20*	-0.05	0.22*	0.37*	-0.03	0.35*	0.34*

Table 27: Correlation coefficients between the extracted metrics and number of defects

For project A, there are in total 287 defects recorded, ranging from 0 to 17 defects per class. Total of 514 defects are recorded for project B, with up to 41 defects per class. Most of the classes in both projects have low number of defects. Histograms of the observed data in Figure 20 show that the normal distribution cannot be assumed for the dependent variable. PRM, NBRM, and ZINBRM are used for dealing with count data.

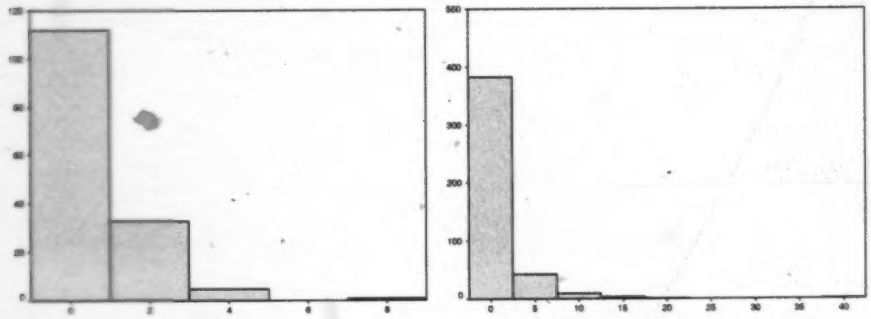


Figure 24:Histogram of the number of defects

The Pareto rule (Ebert and Baisch, 1998) is applicable with respect to the number of defects. Less than 30% of the classes in project A correspond to 80% of the total number of defects, while in project B only 2% of the classes accounts for 80% of defects in the system. The presence of this principle that a relatively small portion of the system consumes most of the effort is closer examined using Alberg diagrams (Ohlsson and Alberg, 1996), shown in Figure 26.

7.2.1. *Extraction of the Models*

For modeling the number of defects, three regression models based on the extracted set of design metrics are used. In these models, the exponent of the regression function is a linear combination of the metrics used as predictors.

The regression function in the PRM can be written in the log-linear form:

$$ln(\mu) = x\beta$$

This suggests that the PRM can be approximated by the linear regression model:

$$ln(y) = x\beta+\varepsilon$$

Since y can also take zero value, it is necessary to add a positive constant c to the dependent variable before taking the log (Long, 1997). Values of c equal to 0.1 and 0.5 are typically used. In this analysis, $c = 0.5$ is used. The resulting regression model is given with:

$$ln(y+c) = x\beta+\varepsilon$$

The parameters of this generalized linear model are estimated using the method of ordinal least squares (OLS). Although potentially biased, OLS estimations can be used for approximation of the statistical significance of parameters in the corresponding PRM (King, 1988).

The stepwise regression method with the log-linear form of the PRM is first applied. All the available metrics are allowed to enter this model. In the stepwise linear regression, independent variables are selected to enter the model based on the p-value. p-value is a measure of the statistical significance, representing the probability the outcome of the analysis is just a result of chance. The lower the p-value is, the higher is the statistical significance of the result. The independent variable with the smallest p-value is entered at each step of the regression, if that value is sufficiently small. Variables already in the model are removed if their p-value becomes sufficiently large. The method terminates when no more variables are eligible for inclusion or removal. In this analysis, p-value 0.01 is used as the entry criteria for a variable to enter the model, and p-value 0.05 is used to remove variable.

The two PRMs (PRM OLS 1 and PRM OLS 2) identified by the stepwise method for both projects are based on RFC and DIT as predictors. Although colinearity between some of the metrics is relatively high, the two metrics in the identified models (RFC and DIT) are not highly correlated (r = 0.02 and 0.03 for two datasets). In both models, parameters associated with RFC have positive, highly significant values.

	Project A			**Project B**		
	Intercept	*RFC*	*DIT*	*Intercept*	*RFC*	*DIT*
PRM OLS 1	-1.63*	0.64*		-0.42*	0.01*	
PRM OLS 2	-0.09	0.01*	0.07	-0.36*	0.02*	-0.04

Table 28: Coefficients of the PRM with statistical significance

Using the results given with the PRM OLS 1 and PRM OLS 2, the same pair of metrics (RFC and DIT) is used for building the other models. Instead of the OLS, the more general ML method is used for estimation of the PRM, NBRM, and ZINBRM. In the ZINBRM, the logit model is used for prediction of zeroes in combination with the NBRM.

The comparison of the resulting univariate (with extension 1) and bivariate (with extension 2) models with respect to the correlation coefficient *r*, dispersion parameter α, and Relative Square Error (RSE) is provided in Table 29.

		Univariate			**Bivariate**		
		r	α	*RSE*	*r*	α	*RSE*
Project A	PRM OLS	0.401	13.47	0.79	0.416	13.24	0.79
	PRM ML	0.713	2.07	0.79	0.749	1.85	0.79
	NBRM	0.424	0.51	0.79	0.467	0.42	0.80
	ZINBRM	0.782	0.46	0.80	0.783	0.39	0.79
Project	PRM OLS	0.280	0.47	9.15	0.287	0.65	9.12

		Univariate			Bivariate		
B	PRM ML	0.240	0.32	9.57	0.232	0.48	9.84
	NBRM	0.225	0.31	11.86	0.203	0.18	19.38
	ZINBRM	0.229	0.24	11.82	0.206	0.16	19.24

Table 29: Resulting models

7.2.2. Analysis of the Results

Two reasonably good PRMs based on RFC and DIT (univariate and bivariate) are identified by the stepwise regression. These models in the log-linear form are given with:

$$ln(NumberOfDefects + 0.5) = \beta_0 + \beta_1 RFC + \beta_2 DIT$$

Where β_i parameters are zero for metrics that do not enter a specific model.

Corresponding NBRM and ZINBRM are built using the ML for estimation of parameters. Results in Table 29 suggest significant differences between the models and some quantitative improvements in compared to the baseline PRM OLS models.

First, a high level of overdispersion is present in the PRM OLS models. This result is not surprising for the software metrics data. However, overdispersion is significantly lower with ML estimators. The other result of the ML method is increased correlation between the model prediction and the observed data.

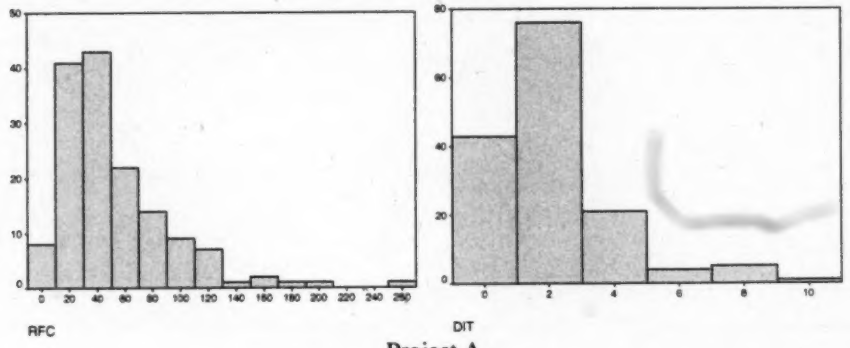
Models based on the NB distribution show even higher capability of dealing with overdispersion. This improvement is partially the result of increased probability of low and high values of the dependent variable with the NB distribution.

Although the NBRM deals with the overdispersion more successfully than PRM, the disadvantage is a lower resulting correlation with the observed data.

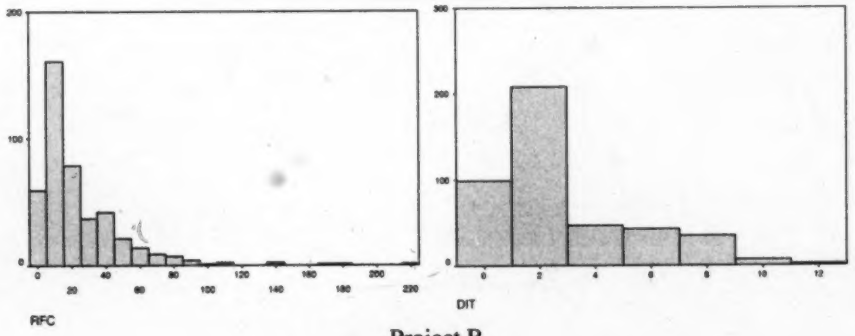
ZINB regression models result in the highest correlation coefficient and the lowest dispersion parameter. These models incorporate the capability to successfully predict zero values with the ability of the NB distribution to account

for overdispersion. Improvements achieved by the ZINBRM come with the expense of doubling the number of model parameters compared to the original NBRM.

The resulting bivariate models in Table 29 show slightly better performance than univariate models. The additional variable enables bivariate models to better fit the observed data. However, this improvement is modest and depends on many factors, such as the underlying dataset.



Project A



Project B

Figure 25: Histograms of RFC and DIT

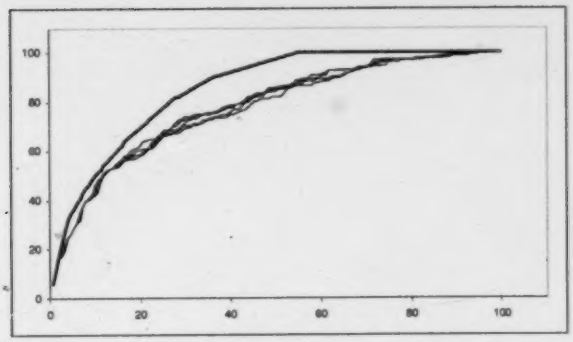
Since the metrics are non-negative, positive values of model parameters suggest positive influence of the predictors to the dependent variable. The link between the dependent variable and predictors is exponential in the analyzed regression models.

All resulting models suggest high influence of communication between classes measured by RFC to the dependent variable in the analysis. The additional predictor, the measure of inheritance DIT, does not significantly increase the performance of the bivariate models compared to the univariate RFC-based models. In addition, model parameters associated with DIT have much lower statistical significance than RFC-related parameters.

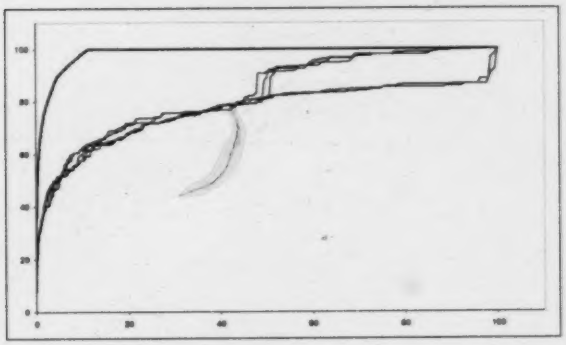
Relatively narrow range of DIT also limits the influence of this measure to the resulting models. RFC is a measure with a wider range of values. In this study, RFC ranged from 5 up to 250. Clearly, the higher the communication of a class with other classes, the higher will be the need for modifications in that class.

As mentioned, Alberg diagrams are used to graphically compare the ability of different models for criticality prediction of the modules in the software system (Ohlsson and Alberg, 1996). Figure 26 shows Alberg diagrams for the models employed on the dataset in this study with respect to the number of defects., The lines representing the resulting models in the graph are close to each other and partially overlapped, evidencing very similar performance of the models in the criticality prediction.

The line representing the observed number of defects shows that less than 30% of the classes cause 80% of the total number of the defects in the system. Alberg diagram shows that both univariate and bivariate models have relatively good performance in prediction of the defect-prone classes. Using any of those models, less than 50% of the classes can be identified that cause 80% of the defects in the system. Performance of the models ranges from identification of 43 % for bivariate models to 45 % for univariate models.



Project A



Project B

	Project A		Project B	
	Univariate	*Bivariate*	*Univariate*	*Bivariate*
PRM OLS	45	43	48	48
PRM ML	45	43	48	44
NBRM	45	44	47	45
ZINBRM	45	44	47	45

Figure 26: Alberg diagrams for the models and predicted percentiles of the classes with 80% of the defects in the system

As mentioned, RFC is a measure of communication between classes defined as the number of internal and external methods that can be executed in response to a

message received by an object of the class. Clearly, a large number of methods, potentially invoked from a class, increases the complexity of the class and requires greater level of understanding and effort in development and debugging of the class.

Depth of inheritance tree represents the position the class has in the inheritance hierarchy, i.e. the maximum distance from the root node when multiple inheritance is supported, i.e. in C++. Although the deeper position of the class in the hierarchy increases its potential for reuse through inheritance, at the same time the increased number of inherited methods makes it more difficult to predict the behavior of the class. It can be beneficial to keep the level of inheritance relatively low, at expense of compromising the potential of reuse through inheritance, in order to enable easier testing and understanding of design and implementation.

While some of the object-oriented design metrics tend to change through the evolution of the project from the design phase to implementation, DIT is one of the measures determined by the architecture of the system and it typically stays unchanged in this process. Consequently, the depth of inheritance tree information can be used relatively early in the software development process.

In order to quantify applicability, effectiveness, predictive ability, and capability of the models, they are ranked according to correlation coefficients, overdispersion and RSE in Table 30. Models with the same performance with respect to a specific factor share cells in the table.

Effectiveness of the model can be expressed in terms of correlation of its estimations with the observed data and in terms of RSE. PRM OLS and PRM ML models show best performance with respect to these criteria on both datasets.

Part of the capability of the models to explain the empirical data is attributed to their ability to account for overdispersion and occurrence of zero counts. With respect to these criteria, using dispersion parameter α as a measure, ZINB and NBRM perform best on the two datasets.

Alberg diagrams can be used to assess both effectiveness of the models and their predictive ability. Alberg diagrams applied to data in this analysis demonstrate

only minor differences in performance of the models. Alberg diagrams support higher effectiveness and predictive ability of bivariate models in comparison to univariate models.

		Correlation		Overdispersion		Square Error	
		Project A	*Project B*	*Project A*	*Project B*	*Project A*	*Project B*
Univariate	1	ZINBRM	PRM OLS	NBRM	ZINBRM	PRM OLS	PRM OLS
	2	PRM OLS	PRM ML	PRM ML	NBRM	PRM ML	PRM ML
	3	PRM ML	ZINBRM	PRM OLS	PRM ML	NBRM	ZINBRM
	4	NBRM	NBRM	ZINBRM	PRM OLS	ZINBRM	NBRM
Bivariate	1	ZINBRM	PRM OLS	ZINBRM	ZINBRM	PRM OLS	PRM OLS
	2	PRM ML	PRM ML	NBRM	NBRM	PRM ML	PRM ML
	3	NBRM	ZINBRM	PRM ML	PRM ML	ZINBRM	ZINBRM
	4	PRM OLS	NBRM	PRM OLS	PRM OLS	NBRM	NBRM

Table 30: Ranking of the models with respect to the applied criteria

7.3. Related Studies and Discussion

Table 31 in this chapter presents an overview of the related work in this area and comparison to the study performed in this study.

Having these papers focused on different dependent variables using different explanatory metrics at the same time, it is not possible to make strict comparison of the results or to make strong generalization. However, all the dependent variables used in these papers are trying to explain various aspects of software development effort using design measures.

Although the dependent variables are heavily skewed and non-normally distributed (Li and Henri, 1993; Chidamber *et al.*, 1998, Ronchetti and Succi, 2000), only Briand and Wüst take this aspect into account by applying Poisson regression. For the same reasons, Basili *et al.* (1996) decide not to use defect count or density but binary dependent variable representing probability of defects for a class.

Furthermore, Li and Henri (1993) do not clearly specify the form of the model they use. Set of metrics used in the papers is also not uniform. Ronchetti and Succi (2000) use only subset of CK suite available to them, while Briand and Wüst (1999) use variety of other metrics in combination with the CK suite,

Considerable influence of RFC to the dependent variable was reported by Basili *et al.* (1996), while Chidamber *et al.* (1998) find CBO and LCOM and Li and Henry (1993) find CBO as dominant factors in explaining independent variables in their studies. Paper by Ronchetti and Succi (2000), also performed in the telecommunication domain, reports NOM as a dominant factor in explaining software size as a proxy of development effort.

This study contributes to the existing research on this subject by focusing on specific application domains using novel statistical methods applied to data from industrial projects in telecommunication and commercial application domains.

Study	**Li and Henri, 1993**	**Basili *et al.*, 1996**	**Chidamber *et al.*, 1998**	**Briand and Wüst, 1999**	**This study**
Environment	Industrial	University	Industrial	University	Industrial
Lifecycle phase of the independent variable	Design & code	Code	Design & code	Design & code	Design & code
Programming language	Classic-ADA	C++	C++	C++	C++
Application domain	User interface and scientific	Information system	Financial application	Music editor	Real-time telecom and commercial
Size	2 projects	8 projects	3 projects	1 project	7 projects
Dependent variables	Number of lines changed during maintenance	Number of faults	Productivity, rework effort, design effort	Development time as a proxy for effort	Number of defects for a class

Study	Li and Henri, 1993	Basili *et al.*, 1996	Chidamber *et al.*, 1998	Briand and Wüst, 1999	This study
	as a proxy for maintenance effort			effort	
Statistical analysis	Parametric linear regression	Logistic regression	Stepwise linear regression including dummy variables	Poisson regression and regression trees	Poisson, NB, and ZINB regression
Conclusions	No other CK metrics but CBO influence significantly the dependent variable	NOM, DIT, CBO, RFC significantly influence the dependent variable	High values of CBO and LCOM significantly influence the dependent variable	Size measures can be used as good predictors of effort	Overall RFC has dominant statistically significant effect to the dependent variable
Other considerations	N/A	The cross-correlation of the CK metrics are low	DIT and NOC assume low values. CBO, NOM, and RFC are highly correlated; the other correlations between metrics are low	Coupling measures do not substantially improve quality of the models	LCOM, CBO, and DIT also have significant influence to some of the analyzed projects

Table 31: Related studies

8. Conclusion

Quality is a key element in success of any software product. A proper characterization and understanding of this process is essential in achieving higher quality software. Models based on software metrics are employed in this study to provide a way of quantitative management of software quality.

A great part of effort and time in the development of software products is spent on servicing demands for modifications in the behavior of the systems. This study builds a systematic and replicable framework for the analysis of SRs.

Design of an object-oriented software system offers a substantial amount of information about the system even before any coding has started. This study empirically validates the ability of the CK object-oriented design metrics suite to be used for identification of the critical classes that consume most of the development effort and resources and need special attention in the development and testing activities.

This study proposes innovative methods for prediction and description of defect behavior of the classes in object-oriented systems, and investigates possibilities for better resource allocation in the process servicing requests for modifications in such systems. The existing and new software reliability growth models are also adopted to characterize the timings of service requests and help in resource allocation. Novel statistical methods are applied to deal with the peculiarities of the software engineering data, such as non-normally distribution, overdispersed count data on the absolute measurement scale, and high occurrence of zero counts. This approach is validated on different datasets from industrial environment in real-time telecommunication and commercial application domains.

Results of the analysis indicate that models based on Poisson and negative binomial distribution successfully explain the excessive variability in the data and can be used to identify approximately 50% of the classes causing 80% of the defects in the system. Measure of communication between classes, i.e., response

for a class, shows a good potential as a predictor for this purpose over the projects analyzed in this study.

The proposed framework for an accurate description of the various aspects of SRs occurrence resulted in a multimodel approach based on software reliability growth models, gamma analysis, and parametric linear regression. A Monte Carlo simulation is performed to assess the sensitivity of the proposed models to the imprecision of the input data, typical when people participate in the data collection process. Results indicate that the multimodel approach is recommended for most reliable results in analysis of service requests.

The work performed in this study can be continued using new available datasets and applying the framework for empirical investigation established in this study.

Some of the proposed techniques could be used for building tools to support decision-making in the software development process. A prototype of such a tool based on the proposed multimodel approach for analysis of SRs is currently being considered.

Acknowledgments

This study was supported in part by Nortel Networks, the University of Alberta, the University of Calgary, the government of Alberta, and the Natural Science and Engineering Research Council of Canada for supporting this research partially.

Bibliography:

Albrecht A.J. and Gaffney J. (1983) "Software function, source lines of code and development effort prediction," *IEEE Transactions on Software Engineering*, 9(6), pp. 639-48

Bansiya J. and Davis C. (1999) "Design and code complexity metrics for OO classes," *Journal of Object Oriented Programming*, 12(1), 35-40

Basili V.R., Briand L.C., Melo W.L. (1996) "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, 22(10)

Basili V.R. and Rombach H.D. (1988) "The TAME project: Towards improvement-oriented software environments," *IEEE Transactions on Software Engineering*, 14(6), pp. 758-73

Basili V.R. and Selby R.W. (1991) "Paradigms for Experimentation and Empirical Studies in Software Engineering," *Reliability Engineering and Safety*, 32, 171-191

Basili V.R., Selby R.W., and Hutchens D.H. (1986) "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, 12(7), July

Basili V.R. and Weiss D.M. (1984) "A method for collecting valid software engineering data" *IEEE Transactions on Software Engineering*, 10(6), 728-738

Booch G. (1994) *Object-Oriented Analysis and Design With Applications*, Object Technology Series, Addison-Wesley

Briand L.C., El Emam K., and Morasca S. (1996a) "On the Application of Measurement Theory in Software Engineering," *Journal of Empirical Software Engineering*, 1(1)

Briand L.C., Morasca S., Basili V.R. (1996b) "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, pp 68-86

Briand L.C., Daly J., Porter V., Wüst J. (1998) "Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems," *The Ninth International Symposium on Software Reliability Engineering*, Paderborn, Germany

Briand L.C. and Wüst J. (1999) "The impact of Design on Development Cost in Object-Oriented Systems," Technical report,

http://www.iese.fhg.de/network/ISERN/pub/technical_reports/isern-99-16.pdf

Brooks F.P. (1987) "No silver bullet: Essence and accidents of software engineering," *IEEE Computer*, 20(4), April

Cameron A.C. and Trivedi P.K. (1986) "Econometric models based on count data: Comparisons and applications of some estimators and tests," *Journal of Applied Econometrics*, 1, 29-93

Chidamber S.R. and Kemerer C.F. (1994) "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6

Chidamber, S.R., Darcy D.P., Kemerer C.F. (1998) "Managerial Use of Object-Oriented Software: An Explanatory Analysis," *IEEE Transactions on Software Engineering*, 24(8)

Constantine L.L and Yourdon E. (1979) *Structured design*, Prentice-Hall

Ebert C. and Baisch E. (1998) "Industrial Application of Criticality Predictions in Software Development," *The Ninth International Symposium on Software Reliability Engineering*, Paderborn, Germany

Fenton N.E. and Neil M. (1999) "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, 25(5), 675-689

Fenton N.E. and Pfleeger S.L. (1997) Software Metrics: A Rigorous and Practical Approach, Brooks/Cole Pub Co

Frakes W. and Terry C. (1996) "Software reuse: Metrics and models," *ACM Computing Surveys*, 28(2), June

Freund J.E. and Simon G.A. (1996) *Modern Elementary Statistics*, Prentice Hall

Goel A.L. and Okumoto K. (1979) "Time Dependent Error Detection Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability*, Vol. 28, No. 3, Aug., pp. 206-211

Gourieroux C., Monfort A., Trognon A. (1984) "Pseudo maximum likelihood methods: Applications to Poisson models," *Econometrica*, 52, 701-720

Gibbs W.W. (1994) "Software's Chronic Crisis," *Scientific American*, September, pp. 86-95

Graves T., Karr A.F., Marron J.S., Siy H. (2000) "Predicting Fault Incidence Using Software Change History," *IEEE Transactions on Software Engineering*, Vol. 26, No. 7, July

Gray A.R. and MacDonell S.G. (1997) "A comparison of techniques for developing predictive models of software metrics," *Information and Software Technology*, 39, 425-437

Hossain S., R. Dahiya (1993), "Estimating the Parameters of a Non-Homogeneous Poisson-Process Model of Software Reliability," *IEEE Transactions Reliability*, Dec., pp. 504-612.

Humphrey W.S. (1989) *Managing the software process*, Addison-Wesley

Ince D.C (1994) *ISO 9001 and software quality assurance*, McGraw-Hill

Jackson M.A. (1983) *System development*, Prentice-Hall

Jacobson I. (1992) *Object-oriented software engineering: a use case driven approach*, Addison-Wesley Publishing Co., 1992

Kamiya T., Kusumoto S., Inoue K., Mohri Y. (1999) "Empirical evaluation of reuse sensitiveness of complexity metrics," *Information and Software Technology*, 41 (5), 297-305, March

Kececioglu D. (1991), *Reliability Engineering Handbook*, Vol. 2, Prentice-Hall

Kemerer C.F., S. Slaughter (1999), "An Empirical Approach to Studying Software Evolution", *IEEE Transactions on Software Engineering*, Vol. 25, No. 4, July/August, pp. 493-509

Kemerer C.F. (1987) "An empirical validation of software cost estimation models," *Communications of ACM*, 30, 416-429

King G. (1988) "Statistical Models for Political Science Event Counts: Bias in Conventional Procedures and Evidence for the Exponential Poisson Regression Model," *American Journal of Political Science*, 32, 838-863

Kitchenham B. (1998) "A procedure for analyzing unbalanced datasets," IEEE Transactions on Software Engineering, Vol. 24, No. 4, April

Labovitz s. (1971) "In defense of assigning numbers to ranks," *American Sociological Review*, 36, pp. 521-522

Lambert D. (1990) "Zero-inflated poisson regression with an application to defects in manufacturing," *Technometrics*, 34, 1-14

Li W. (1998) "Another metric suite for object-oriented programming," *Journal of Systems and Software*, 44(2), 155-162,

Littlewood B. (1981), "Stochastic Reliability Growth: A Model for Fault Removal in Computer Programs and Hardware Design," *IEEE Transactions on Reliability*, Dec. pp.313-320

Lloyd C.J. (1999) *Statistical Analysis of Categorical Data*, Wiley-Interscience

Long J.S. (1997) *Regression Models for Categorical and Limited Dependent Variables,* Advanced Quantitative Techniques in the Social Sciences, No 7, Sage Publications

Lyu M. R. (1996), Handbook of Software Reliability Engineering, McGraw Hill.

Pelz D. C. (1985), "Innovation Complexity and the Sequence of Innovating Stages", *Knowledge: Creation, Diffusion, Utilization*, vol. 6, pp. 261-291.

Marchesi M. (1998) "OOA metrics for the Unified Modeling Language," *Second Euromicro Conference on Software Maintenance and Reengineering*, Los Alamitos, CA, USA

Mayer L. (1971) "A note on treating ordinal data as interval data." American Sociological Review, 36, pp. 519-520

McCabe T.J. (1976) "A complexity measure," *IEEE Transactions on Software Engineering*, SE-2(4), pp. 308-20

Metamata (2001) *JavaCC Documentation*, http://download.metamata.com/javaccdocs.zip

Miller B.K., Pei H., Chenho K. (1999) "Object-oriented architecture measures," *32nd Annual Hawaii International Conference on Systems Sciences*, Los Alamitos, CA, USA

Musa J. D. (1984), "Software Reliability", *Handbook of Software Engineering*, C. R. Vick and C. V. Ramamoorthy (Eds.), pp. 392-412, Van Nostrand Reinhold

Musa J., Iannino A., and Okumoto K. (1987), *Software Reliability*, McGraw-Hill.

Nara T., Nakata M., Ooishi A. (1995), "Software Reliability Growth Analysis: Application of NHPP Model and its Evaluation", *Sixth International Symposium on Software Reliability Engineering ISSRE'95*, Toulouse, France

Nesi P. and Querci T. (1998) "Effort estimation and prediction of object-oriented systems," *Journal of Systems and Software*, 42(1), 89-102

Nikora A. P. and Lyu M. R. (1995), "An Experiment in Determining Software Reliability Model Applicability", *Sixth International Symposium on Software Reliability Engineering ISSRE '95*, Toulouse, France

Ohlsson N. and Alberg H. (1996) "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Transactions on Software Engineering*, Vol. 22, No. 12, December

Papoulis A. (1991) *Probability, Random Variables, and Stochastic Processes*, McGraw Hill College Div

Paulk M.C. (1995) "How ISO 9000 compares with the CMM," *IEEE Software*, 12(1), 74-84

Pressman R.S. (1996) *Software Engineering: A Practitioner's Approach*, McGraw-Hill Higher Education

Reyes L. and Carver D. (1998) 'Predicting object reuse using metrics', *SEKE '98. Tenth International Conference on Software Engineering and Knowledge Engineering*, Skokie, IL, USA

Riel A.J. (1996) *Object-Oriented Design Heuristics*, Addison-Wesley Pub Co

Ronchetti M. and Succi G. (2000) "Early Estimation of Software Size in Object-oriented environment – a Case Study in a CMM Level 3 Software Firm," *submitted*

Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Blaha S. (1991) *Object-Oriented Modeling and Design: Solutions Manual*, Prentice Hall

Sommerville I. (2001) *Software Engineering*, Addison-Wesley Pub Co

Succi G., Benedicenti L., Bonamico C., Vernazza T. (1998) "The Webmetrics Project - Exploiting 'Software Tools on Demand'," *World Multiconference on Systemics, Cybernetics, and Informatics*, Orlando, FL

Succi G. and Uhlik C. (1997), "The Compilation of SL, a set-based logic language for generic parallel architectures," *Journal of Programming Languages*, 5, 27-74

Shih T.K., Lin Y-C, Pai W.C., Wang C-C (1998) "An Object-Oriented Design Complexity Metric Based on Inheritance Relationships," *International Journal of Software Engineering and Knowledge Engineering*, 8(4), 541-566

Teologlou G. (1999) "Measuring object-oriented software with predictive object points," *10th European Software Control & Metrics Conference*, Herstmonceux, England

Tian J., Lu P., Palma J. (1995), "Test-Execution-Based Reliability Measurement and Modeling for Large Commercial Software", *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, May, pp. 405-414

Weyuker E. (1988) "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, September

Williams L., Kessler R.R., Cunningham W., and Jeffries R. (2000) "Strengthening the Case for Pair Programming," *IEEE Software*, Vol. 17, No. 4, July/August

Wood A. (1996), "Predicting Software Reliability", *IEEE Computer*, Vol. 29, No. 11, November, pp. 69-77

Yamada S., Ohba M., and Osaki S. (1983), "S-Shaped Reliability Growth Modeling for Software Error Detection", *IEEE Transactions Reliability*, Dec., pp. 475-484.

Yamada S., Ohtera H., and Narihisa H. (1986), "Software Reliability Growth Models with Testing Effort," *IEEE Transactions Reliability*, Apr., pp. 19-23

Yand Y. and Weber R. (1990) "An ontological model of an information system," *IEEE Transactions on Software Engineering*, 16, pp. 1282092